# RE-ROUTING USING CONTRACTION HIERARCHIES IN SOFTWARE-DEFINED NETWORKS

By

Sebastian Miranda

A Thesis submitted to the

Faculty of Graduate Studies

in Partial Fulfilment of the Requirements for

the degree of Master of Science

in the Department of Applied Computer Science

University of Winnipeg

Winnipeg, Manitoba, Canada

October 2020

# Abstract

Nowadays, information services are a necessity for our everyday lives, the necessity of communication and information transmission increases as fast as the number of users and the number of the devices. Especially during times when direct communication is not an option, there is an increase of the demand for these services. Researchers have continuously developed multiple options to cope with the transmission increase but solving all the possible problems seems away from reality as solutions depend on many case scenarios. One of the best approaches is to develop solutions that can cope with specific problems or improve specific areas depending on the necessity of the network.

In this thesis, we propose a theorical approach to routing on computer networks by analyzing the similarities and differences between transportation networks and computer networks. Within transportation networks we focus on road networks to carry out this research. The similarities will allow us to implement (on computer networks) the Contraction Hierarchies algorithm (CH). CH is an interdisciplinary algorithm originally developed for road networks, which can provide us with the elements and logic to optimize specific routing problems in computer networks. To implement CH, we use Software-Defined Networks (SDN). SDN is a computer networks paradigm that separates the Data and Control planes. The Data plane is left to the network devices to distribute the packages, and the control plane is centralized into a Controller. By having a controller with a broad view of the network, we implement CH to optimize route selection.

Once the route is determined, we study the possibility of using the advantages of CH to redistribute traffic in case the network elements suffer from unforeseen circumstances during transmissions. The performance of our algorithm is measured based on its capability to find an alternative route between two nodes (as long as the alternative route is available) and the delay produced on the communication while the route is created. The performance during unforeseen circumstances demonstrates the capacity of our algorithm to maintain the communications and improve the survivability of the network.

# Acknowledgments

First, I would like to express my deepest gratitude to my supervisor Dr. Sergio Camorlinga for his patience and guidance during the development of this work. His advice and support have been an elemental part of my motivation to complete my research.

I want to specially thank my parents Roberto Miranda and Sara Rodríguez, and my siblings Jeronimo Miranda and Valeria Miranda who are the biggest inspiration of my life and encourage me to be a better person everyday. Without them, this work would not be possible.

To my Canadian family Gus Bravo, Paula Hossack, Christopher Hossack and my lovely nieces who welcome me and gave me their support and hospitality during the time I spend on the master's degree.

To my partner Kelly Thornham who gave me her unconditional support and motivation to continue.

Last but not least, to all the teachers and friends who help me during this thesis work.

# Table of Contents

# Table of Figures

# Glossary of Terms

| | |
|---|---|
| ARP | Address Resolution Protocol |
| BFS | Breadth-First Search |
| BS | Bi-directional Search |
| CDN | Content Distribution Network |
| CH | Contraction Hierarchies |
| CPU | Central Processing Unit |
| DDoS | Distributed Denial of Service |
| DFS | Depth-First Search |
| D-ITG | Distributed-Internet Traffic Generator |
| DST | Destination |
| D-to-S | Destination to Source |
| ED | Edge Difference |
| GPS | Global Position System |
| IoT | Internet Of Things |
| IS-IS | Intermediate System to Intermediate System |
| ISP | Internet Service Provider |
| LA | Lion Algorithm |
| LACP | Link Aggregation Control Protocol |
| MAC | Media Access Control |

| | |
|---|---|
| ONF | Open Networking Foundation |
| OSPF | Open Shortest Path First |
| P2P | Peer-to-Peer |
| P2P-CDN | Peer-to-Peer-Content Distribution Network |
| QoE | Quality of Experience |
| QoS | Quality of Service |
| RSU | Roadside Units |
| RTT | Round-Trip Time |
| SDN | Software Defined Networks |
| SPOF | Single Point of Failure |
| SRC | Source |
| S-to-D | Source to Destination |
| SW | Switch |
| TTL | Time To Live |
| V2I | Vehicle to Infrastructure |
| V2V | Vehicle to Vehicle |
| VANET | Vehicular Ad-Hoc Networks |

# Chapter 1

# Introduction

In this thesis, we research the similarities between transportation networks and computer networks. Within transportation networks we focus on road networks to carry out this research[1]. The similarities on both systems will allow us to implement on computer networks one of the ideas used to improve road networks. Road networks algorithms can provide efficient mechanisms to optimize network path selection during network devices failures. In this work, we explore the option of using Software Define Networks to implement our ideas. This chapter sets a general context of our research. First, on Section 1.1, we start with a general background on content distribution networks. Followed by the scope and some limitations to narrow the objectives of our research on Section 1.2. Then on Sections 1.3 and 1.4 we provide a brief description of the motivation and the problems that inspired us to perform this research. Section 1.5 presents the objectives of our research. Finally, on Section 1.6, we describe the organization of the rest of the thesis.

## 1.1 Content distribution background

With the increase of network infrastructure and topologies, new opportunities for information distribution techniques and advertisement arise, we can distribute our applications, information, videos, images, etc., at a higher speed

[1] Road networks is used as an instance of transportation networks throughout the thesis.

regardless of the distance. Organizations with presence in different countries and continents can almost instantaneously observe the situation on different locations and share information according to their needs.

Every time a company or organization is created, there is a need to include computer programs and network infrastructure (web pages, databases, mobile applications and/or desktop applications, etc.). Nowadays businesses that fail to make use of technological advances are left behind or disappear. As companies keep expanding and include more users to their services, they need to implement newer technologies and update the old ones to support more and larger loads of data.

Internet has grown so fast in terms of users, services, and content providers, that different ways to share information are needed. The basic client-server architecture model of an organization with one (or a few) server(s) distributing the content for one or more clients as shown in Figure 1, is not enough anymore for big companies like Google, Facebook and Netflix, who distribute data amongst millions of users around the world every day. Only increasing the number of servers in the same data center, makes close to no difference while distributing the information for millions of users in different locations. Distribution of data is not the only problem faced, distributing different files for each region, different languages, different formats, heavier files such as music, videos, databases, and dynamic content on the webpages, makes this problem even more complicated. e.g. [Tan15] shows the problematic faced by a large organization (Facebook) which manages multiple applications, types of data, multiple servers and services, different configurations and updates for the apps, and multiple languages for different users.

Sending different types of information from one or a few servers to all the clients produces huge amounts of traffic on the network. Researchers have created and implemented many software and hardware solutions to this problem (reduce the traffic while distributing the same amounts of information).

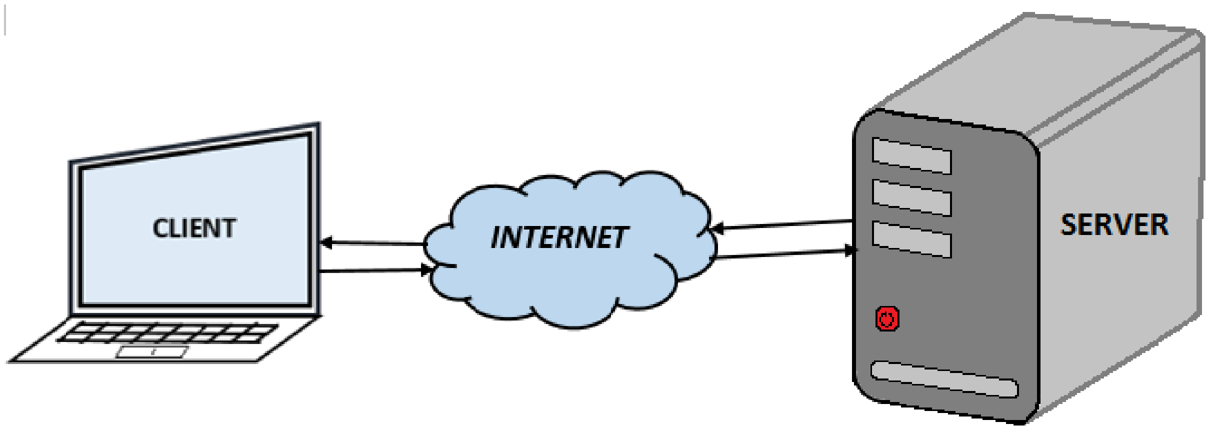


*Figure 1 Client - Server Architecture*

Amongst the most common hardware solutions, we can think of making changes to the network topology. These changes include adding and/or changing network devices with better capabilities, new links between the devices to improve connectivity, and topology changes. We will briefly describe the most common and interesting network topologies we found.

One of the best hardware options for companies with many files but few servers, is sharing data to some clients, and then share it from client to client. This model is known as the Peer-to-Peer model (see Figure 2). The peer-to-peer model (P2P) is a variation of the client-server model where there is no specific client or server, when a node (a device in the network) is receiving information it becomes the client, but when the same node is sharing the information, then it becomes the server that distributes the information to other clients.

This method has some advantages and disadvantages. When the number of users in the network increases also the number of nodes that share the information increases, The increased number of users reduces the number of servers the organization must possess in order to distribute the information and brings high scalability on the system with less cost of transmission. As the organization of the network changes, transmissions use the infrastructure of other networks. This increases the importance of the routing between users and offers multiple paths between them. Nevertheless, when the number of clients is scarce or the clients disconnect in the middle of a transmission, the transmission is interrupted and the node that is disconnected (sharing or receiving the information) will not be available for further transmissions until the node's connection with the network is re-established.

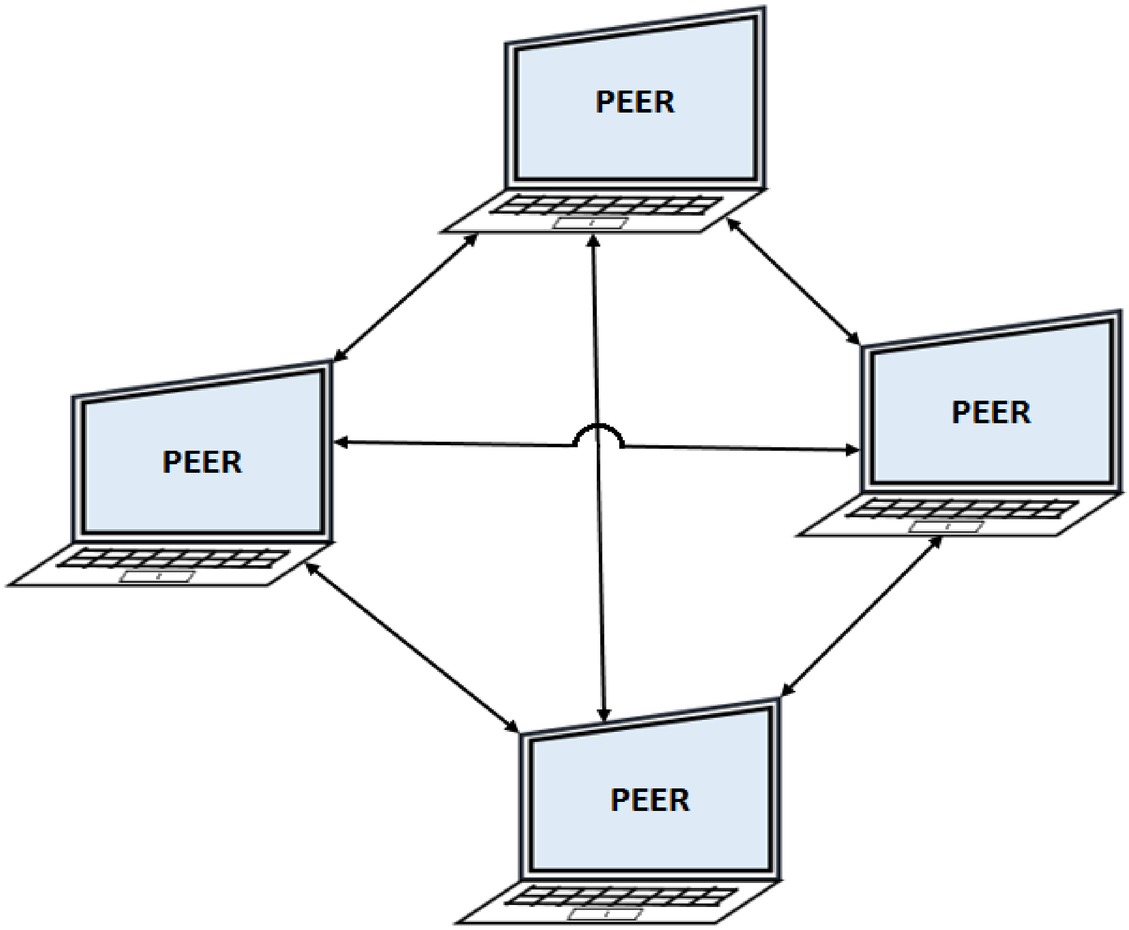


*Figure 2 P2P Architecture*

The peers in the peer-to-peer network can be clients independent from the organization just searching for specific files that they need, and once they get the files, the clients can log out of the system or turn off their computers without previous notice. As we cannot be certain when a client will be available for transmission, we cannot guarantee the availability of the transmitters. The files that a client will choose to download and transmit will be managed by the user which can modify them and send a corrupted version of the same file. If the users decide to delete or move the files from their computers, they will not be available anymore. Finally, the speed available for distribution will be different for each peer, depending (amongst other factors) on the individual client's internet connection and computer capabilities.

A simple peer-to-peer method is a considerable option when the files shared are requested by a large number of clients in the network. Nevertheless, it is not recommended when the files to be transmitted are of high importance or the number of clients is scarce and the few peers are away from each other, the transmission peers could be distributed around the globe and the transmitter and receiver can be separated geographically by large distances, which can produce big delays and lost packages during the transmission. In the case of few peers in the network separated by large physical distances the strongest advantages of the P2P network are lost, and the times of transmission can be larger in comparison with client-server topologies.

A second hardware option to share information with multiple users happen when the organization has different locations with data centers, or the organization can obtain (or pay for) storage services around the globe. As seen in Figure 3, the branches of the organization can contain a server (or a cluster of servers, also known as surrogate servers) to distribute the information with local clients. This method has the following advantages:

- The information is shared in shorter distances relative to the main servers.
- The total amount of clients is distributed among multiple servers.
- The closest available server is selected for the transmissions.
- The network administrator(s) of the company or the maintenance team of the storage service provider company can deal with problems that arise with the servers on the different locations.

This topology is known as Content Distribution Network (CDN).

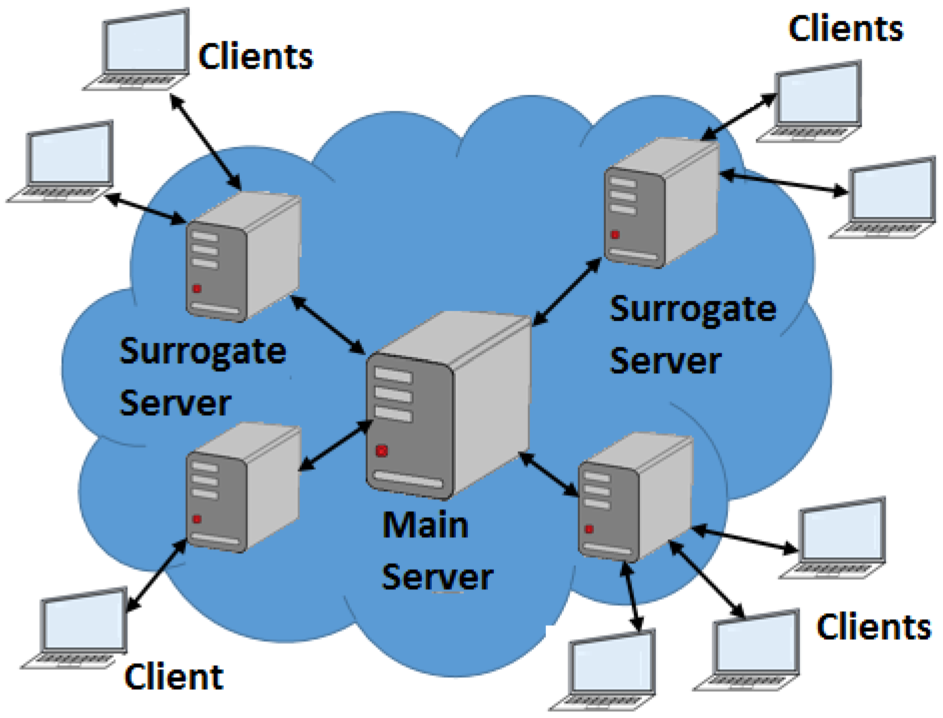


*Figure 3 Content Distribution Network*

A content distribution network consists on many servers or clusters distributed in “zones” around the area of service. These servers can distribute the information to multiple clients in each zone faster and with less errors than the main server because the zones are geographically closer to the clients than the main server of the organization. These short distances between the clients and servers mean less jumps of the packages in the network, less traffic generated through the internet, and less clients (bottlenecks) on the main server. These zones are distributed strategically depending on different factors that benefit the distribution of the information to the clients, such as:

- If the number of clients in a zone (a city, country, or continent) is too large, more servers or different traffic management is required.

- If the amount of traffic generated by several users in a geographic area is considerable as to redirect it to a closer server instead of moving it larger distances to the main server.
- If the allocation of a new surrogate server can reduce the delay time and reduces the packets lost during transmissions.
- The content to be distributed to the clients is different from zone to zone.
- The distance between the clients and the main server is too large compared to that of other branches within the organization.
- Prevent the main server to become a Single Point of Failure (SPOF[2])

Note that not all these cases mean that we must add a server in that zone. As explained in other research works, routing protocols, server distribution, and load balance play an important role while transmitting data. Nevertheless, the concept of locality with the clients can greatly improve the Quality of Service (QoS).

The clients in a CDN can contact the server that is closer to them and obtain the necessary information faster than contacting the original server. This method may seem the easiest way to distribute the information, but there are many factors to be considered, for example:

- Internet Service Providers (ISP) routing policies and connections to other networks
- Locating the closest server to the client (sometimes made by Domain Name Servers)
- Synchronization of the branch servers with the main content provider

[2] A Single Point of Failure is a part of the system that if it fails, will stop the whole system from working.

- Number of clients that can connect to a server or cluster without affecting other services or without generating a bottleneck.
- Types of services provided by the different servers (match the service required by the client)
- Cache server policies employed by the network administrators

Examples of how these factors can affect the QoS are found in [Alc14] and [Kri09]. D'Alconzo et al show a real case scenario where cache selection policies in a large CDN directly affect how users perceive a drop in the transmission rates. Krishnan et al show another example in which routing policies greatly increase the round-trip time (RTT) of packages sent, in some cases up to 300-400 ms (even on packages sent to locations inside of the same country). Although these examples are not recent and probably were already solved, they show the importance of good management in networks.

One of the best examples of a CDN is explained in [Nyg10], [She05], and [Dil02]. In these research papers the authors explain the importance of CDNs, the extension of the Akamai network, how the system works and how it helps multiple internet applications at the same time to improve user's QoS, and offers them acceptable levels of performance, reliability, and cost-effective scalability.

As we notice, both methods mentioned above (P2P and CDN) have their own advantages and disadvantages which lead to new efforts and topologies to solve the problems involved in content distribution and improve the quality of service (QoS) using the advantages of both methods. In [Xu17] the authors develop a method which uses the advantages of the CDN infrastructure without using surrogate servers by recruiting volunteers (regular internet users or any operator with a suitably connected host) who contributes with their machines and internet connection to create a content delivery marketplace (as the authors named it). In this marketplace, volunteers can sign up with multiple content providers to distribute the information via these users who would be compensated in some fashion for their services. P2P networks often use these kinds of compensations in order to motivate the users to stay connected and keep distributing the information.

Another topology often found and reviewed by researchers in which both topologies (P2P and CDN) are merged into one is known as P2P-CDN (the name varies depending on the research, but the main idea is similar, see Figure 4). The P2P-CDN method uses the advantages of both topologies, having a main server which distributes the information to edge servers, then other peers help the edge server in the process of distributing the information to different clients.

Using the main and surrogate servers as the core of the topology improves the availability and locality of the system and having a P2P structure to complement the CDN makes the distribution around the edge servers faster and more efficient. Also, to change the topology from a CDN to a P2P-CDN topology takes few changes when it comes to hardware. Researchers agree that as this paradigm solves some of the problems combining the advantages of both technologies. On the downside, P2P-CDN also adopts some of the disadvantages of using both.

- The necessity of a software capable of coordinating and distributing data using both topologies at the same time.
- The availability of the P2P nodes.

- Update the data version being distributed in the network and making sure that the files are not corrupted.
- Costs related to pay or to maintain different servers in different locations (although is being reduced by the P2P structure, it is still present), etc.

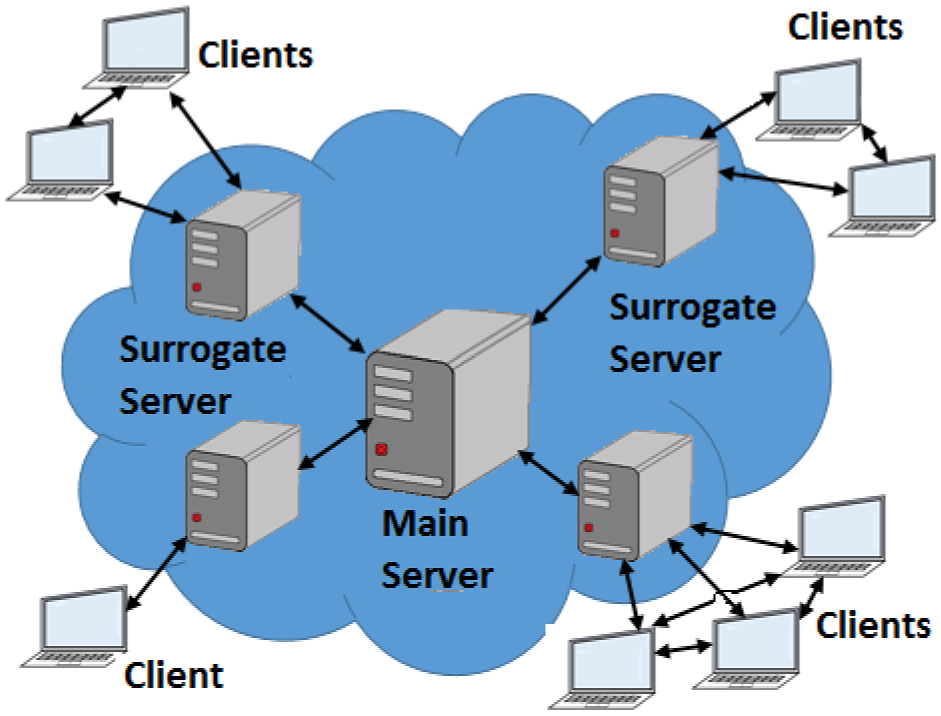


*Figure 4 P2P-CDN*

One clear example of the hybrid CDN-P2P topology is [Ha17] in which the authors test this architecture for live video streaming. First the content creator uses an audio/video device to create the content which is sent to a server called origin server. The origin server's purpose is to hold the chunks in which the video is divided and to distribute the video files to surrogate servers (CDN part). Then the surrogate servers distribute the files amongst the users. Finally, in the lowest level of the topology, users can interact with each other to distribute the files they have received and inform other users of the chunks they have, in order to trade for others, they are missing (P2P network). This method reduces the workload from the surrogate servers and even more from the origin server.

More transmission protocols and topologies are designed day by day, focused on decreasing the transmission delays for the clients, improve the Quality of Service, reduce the costs of transmissions and hardware costs, improve the utilization of the resources, reduce the energy utilization, etc. These optimizations help to greatly reduce and optimize the internet traffic, but as time passes, the internet traffic produced by new apps, new devices, and new users also increases and new optimizations are required.

In 2011 the Open Networking Foundation (ONF) introduces the Software-Defined Networks (SDN). SDN is defined by the organization as, “Decouples the control and forwarding planes enabling network control to be directly programmable”. See Figure 5 for more details on the basic SDN infrastructure. Complementary to SDNs the ONF in 2012 introduces the OpenFlow protocol which is the “First standard communications interface defined between the control and forwarding layers of an SDN architecture”.

SDNs and OpenFlow allow the control plane to be separated from the network devices and managed separately by a server (or servers) known as controller, which can easily be programed by the network administrators. SDN usage is still increasing nowadays. The importance of SDN in this work is the implementation of the management capabilities that can help mitigate some of the problems on package transmission and route planning which will be used and explained during this work.

On the software solutions to effectively distribute information through the network we can find multiple routing protocols. In this chapter we only describe a few commonly used link layer protocols used to find the routes between hosts in the network. We choose these protocols due to the importance of the functionality and relation to this research.

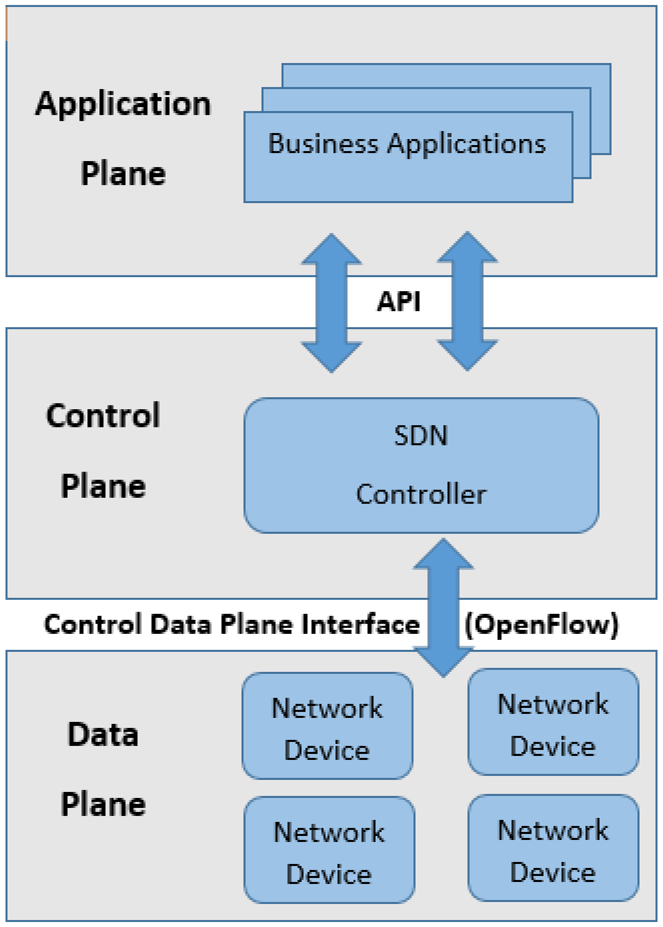


*Figure 5 Software-Defined Network Architecture*

The first protocol to mention is the ARP protocol. This protocol is used to communicate between hosts in the local network. ARP sends request and response messages through the network nodes. These messages contain important information needed to communicate between nodes. Network elements use the broadcast address to transmit the request until the message reaches the destination and then an ARP reply message is issued. This ARP reply contains the MAC address of the destination host. If the ARP communication between each other is successful, then they can begin the communication by using other protocols.

Aside from the simplicity and effectiveness when properly implemented, some problems with the ARP protocol are:

- The traffic produced by the broadcasts and the inability of the network to identify if the request has been received before. This means that loops between network devices produce multiple unnecessary packages and the set of the TTL (Time To Live) becomes important to avoid problems.
- The network elements work as individuals and are uncoordinated.
- The whole process must be repeated every time a new connection must be stablished. This problem also leads to high delay times and interruptions on the communication if problems arise.

Two of the most common protocols used nowadays to find routes between hosts in the network are Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS) [Tei06]. These protocols use network metrics or weights to implement Dijkstra’s algorithm. Dijkstra’s algorithm calculates the shortest path between nodes on a weighted graph. Each router in the network uses multiple messages to communicate to other devices. According to the response of the messages, the network devices calculate the shortest paths in terms of link weights to every other router and builds its own forwarding table.

OSPF and IS-IS contemplate internal network problems during transmissions. According to [Tei06] these protocols use a series of four events to deal with routing events.

1. Detection: in case of equipment failures, other routers can detect them by receiving an explicit alarm or by detecting consecutive keepalive messages.
2. Propagation: after detecting the failure, routers generate link-state advertisement messages to inform other routers about the change.
3. Path re-computation: after receiving a message indicating changes in the network, routers recompute their best paths to all other routers.
4. Forwarding table update(s): some routers may contain more than one routing table, one for each network card.

Although the Dijkstra's algorithm is comparatively a simple and effective algorithm, and the network elements in OSPF and IS-IS present some coordination and communication elements, the communication and updates amongst network elements can be further improved.

Despite all the new protocols, topologies and advancement on the area, the workload on the networks, number of clients, types of content, and application requirements keep increasing day by day. Although there is no specific method or topology to safely and effectively cope with all the routing events, we can offer multiple ideas to solve specific problems.

## 1.2 Research scope

Even though the whole internet is often simplified into one single element (as in Figure 1), it consists on multiple networks (Autonomous Systems) with thousands of servers and network devices interconnected but managed by different companies. Due to the division of the management and specific characteristics of the network, each network deals with their own challenges.

Based on the level of connectivity, network routing protocols are different for internal networks or intradomain routing (e.g. inside the network of an internet service provider) and external communication or interdomain routing (e.g. transmission between multiple service providers). Although very often packages travel from one network to another, for research purposes focusing on the whole internet infrastructure is out of the capabilities of this work. To reduce the scope of this research, we will focus on Intradomain Routing (internal network communication).

Under the benefits of using SDNs listed on [Wyt14] this research goes on the Network Administration category[3], and following the 4-step problem solution from Section 1.1, our algorithm copes with the path re-computation and the forwarding table updates.

Finally, [Yu17] divides the survivability mechanisms into:

- Proactive: different types of proactive mechanisms provide extra bandwidth during resource allocation or evaluate the probability of failures before these happen. Proactive mechanisms then use the

[3] The benefits of SDN for administration are: Administration of the network, traffic distribution according to the network owner policies, and energy saving.

resources and information previously obtained to prevent and recover during unforeseen circumstances.

- Reactive: these types of mechanisms seek for backup bandwidth only after the failure has happened.

Although we acknowledge the importance of proactive mechanisms and their effectivity while mitigating unforeseen circumstances, we are also aware that not all network problems can be prevented. Also, to provide a more general solution, in this research the proposed mechanism falls on the reactive category.

## 1.3 Motivation

Technology and computer science are used to improve the quality of our everyday lives. One of the areas that has greatly benefited from computer science and network development is road networks. Information about real time traffic, traffic patterns, weather, transit limitations (slow streets e.g. pedestrians cross, hospitals, schools, streetlights, roadblocks, accidents), street conditions, etc., can be provided to drivers by using television and radio for traffic news, social networks, sensors, cameras, GPS, etc. While network technologies like ad-hoc networks, SDNs, or internet of things, are used to obtain real time information on road networks, we believe that the same way that technology and networks help on the improvement of road networks, the experience obtained while tracing everyday routes can be used to improve aspects of routing in computer networks.

Despite the multiple possible events that can obstruct roads in road networks, we believe that sufficient work has been done to prevent or reduce the impact of these events. Especially in road networks there are multiple studies that propose the use of protocols and technology to improve traffic. The variety of research include the structure of the roads and city planification which can be related to topology planification on computer networks, and implementation of protocols and technologies to improve the traffic planification. In Chapters 2 and 4 we will explain in detail the logic behind this comparison.

## 1.4 Problem Statement

Although network topologies and protocols are designed and refined to improve transmission and avoid failures, we still do not have the capacity to detect and solve all possible threats to the network. Some of the unforeseen events might be related to maintenance or inconsistencies of the network paths. The capacity to recover from failures often involves long periods of delays since the detection to the re-routing of the network devices. During this recovery period, packets may be dropped due to invalid routes [Kva09]. A disruption lasting a few hundred milliseconds is long enough to interrupt a phone conversation or a video game, and other applications such as web transactions are visibly affected by disruptions lasting a few seconds.

By implementing an SDN, we bring multiple benefits to the network. First, we expect to reduce the processing workload on the network elements. By shifting the route processing to the controller, the network elements can concentrate on the transmission of the data. By knowing the state of the network, the controller can use the information from other transmissions to effectively find the packages destination. Finally, by concentrating the administration on the controller, the reaction to network events can be optimized. Following the previous steps to solve unforeseen events (detection, propagation, path re-computation, forwarding-table update), after detecting the failure of the device, we can simplify the propagation, reduce the re-computation to the controller element, and update the routes as required.

As the effectivity of SDNs depends on the programmability of the controller, as expressed on Section 1.2, this work will be focused on providing an alternative for Intradomain Routing. This alternative is a reactive experimental method based on the inherent similarities between both road and computer networks. In comparison to other protocols like Open shortest path first (OSPF) or Intermediate System to Intermediate System (IS-IS), our method uses the capabilities of SDNs to provide a broad view of the network devices connected to the controller. The connection of the network devices to the controller copes with the individuality of the elements on the network. In comparison to other SDN researches, we believe that the design of our algorithm based on road networks improves the chances of working fast while the network escalates. Other factor that improves the functionality of the algorithm is the creation of the shortcuts that improve the routing as the execution progresses.

## 1.5 Objectives

Many techniques are being used to handle the packages in a network, depending on the logic followed by the route selection algorithm. With the implementation of SDNs, new opportunities for network administration arise.

Constantly new ideas for the algorithm’s logic are being obtained or improved by following an observation process, based on the infrastructure of a specific network, examples found in nature, or similar processes from other disciplines.

These algorithms provide the user with the logic of how to handle a process found in real life or other disciplines and examples of how these situations are solved. By analyzing the elements of the system and their interactions we can find similar agents and variables on the problem we are facing, then take the examples found in these systems to program an algorithm with similar characteristics which can be adjusted to solve our problem.

The objective of this work is to further improve the content distribution algorithm used on a Software-Defined Network’s environment. This improvement will be focused mainly on the availability and survivability of the network tested while some errors occur on the nodes. We will implement an adapted version of the Contraction Hierarchies algorithm along with a re-routing extra module to improve the availability and efficiency of the network routes. The Contraction Hierarchies (CH) algorithm is an algorithm used on road networks with similar characteristics to our environment which produces an order and shortcuts on the topology. CH will allow us to achieve our availability goal and prove that the similarities of both, the road networks and computer networks, makes possible the collaboration between both areas. The program improvement consist on an extra module to redirect the routes when availability issues are present on the nodes, by using the previously calculated shortcuts. The final goal is to prove that by using the programmability of SDNs along with CH, adding a re-direction module, and by re-calculating the new routes in a local way into our SDN server, we can reduce the delays during unforeseen events. By using an interdisciplinary algorithm, we can generate a logic to calculate and re-calculate desired routes within computer networks, and even save the connection with just a small delay for the re-routing when pre-established routes face transmission issues.

The contributions of this research are as follows:

- First, we will perform an analysis of how computer networks export technology to other disciplines (i.e. transit networks), and how computer networks also learn from the experience obtained from other areas. This will become the foundation for adapting an optimization algorithm to our computer networks (Chapters 2 and 3).
- After the comparative analysis of transit and computer networks, we implement an SDN with an interdisciplinary algorithm to administrate traffic and take advantage of the centralized administration capabilities for the traffic distribution. While doing the implementation of the algorithm, we analyze its performance on multiple networks (Chapters 4 and 5).
- Finally, we will demonstrate that with the logic obtained on the previous objective, separating the control plane from the data plane, and by using the shortcuts and hierarchies methodology, we can implement an extra module to improve not only the survivability but also the scalability of the network while tracing routes.

## 1.6 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2 we present a brief literature review and discuss previous research to provide the appropriate context and comparative information. In Chapter 3 we describe the general environment needed to implement our ideas, as well as the programs used to test it. In Chapter 4 we make a comparative analysis of road networks and computer networks and give an introduction to the Contraction Hierarchies algorithm used to solve the path selection process problem and the reasons to select this algorithm for this work. Also, in Chapter 4 we introduce the pseudo algorithms and the reasons to select them. In Chapter 5 we report the implementation results of the algorithms from Chapter 4 and perform the analysis of the results. Then, based on the analysis, we propose the functions of the added module to improve the survivability of our proposed environment. Chapter 6 summarizes the case scenarios we use during our tests and explains the algorithms we propose to solve them. Then, we show the results of our experiments and an analysis of our experiments. Finally, Chapter 7 concludes with the contributions of this research, our conclusions, and suggestions for future works.

# Chapter 2

## Related Works

In this chapter, we provide an overview of the background material and introduce previous research theories and algorithms related with those of our research. First, in Section 2.1, we explain the importance of the SDN architecture and present some of the related works that use this paradigm to improve content distribution, path selection, and other important areas in networks. In Section 2.2, we mention another work who shares the same view that vehicular[4] networks and computer SDNs can work together to solve common problems. Finally, in Section 2.3, we will focus on the importance of Mininet for research purposes. Mininet is the open source network simulator tool selected for this work due to its characteristics and programmability suitable for this work. This section reviews some of the works that focus on Mininet's performance to understand some of the pros and cons of using this simulator from the perspective of other research.

## 2.1 Software-Defined Networks

According to the Open Networking Foundation (ONF) [Onf12], one of the reasons to re-examine traditional network architectures is the increment of mobile devices and its data transmission. This fact is supported by CISCO in

[4] Vehicular network is a different area (than road networks) in Transportation Networks that provide examples on vehicles communication and information gathering. This work uses some of the vehicular network ideas presented on roadside units and information gathering using multiple types of networks and infrastructure.

the document Visual Networking Index [Cis19] in which the mobile data traffic forecast calculates the traffic of 2017 in 12 exabytes per month (only on mobile devices), with an increase to 29 exabytes per month for the year 2019, and an estimate increase to 77 exabytes each month in 2022. This is only the increment of mobile services, other areas such as video streaming are rapidly increasing too. Forbes' statistics about the success of the video streaming company Netflix in [For18] estimate an increase from 81.52 million households subscribed in 2016 to 114.89 million households subscribed in 2020. In [Cis19b], the global IP traffic forecast estimates an overall traffic increase to 396 exabytes per month in 2022, more than three times the traffic on 2017 (122 exabytes per month). These numbers were obtained before the unforeseen incidents on 2020 which increase the necessity of network services and their improvement.

This huge increase in the network demand requires constant updates in the network infrastructure (which increases the expenses on new devices) and a change in the information distribution methods.

There are many proposals to improve distribution depending on the process(es) to be optimized. Software-Defined Networking (SDN) as previously defined, is a network architecture that decouples the control plane from the data plane. SDN eases and centralizes the programing functions of the network elements. In the SDN model, the network devices focus on forwarding the packages (data plane), while the control plane is delegated to a central server known as controller. Nevertheless, SDN is not the only option to optimize content delivery. In the survey made by Jia et al [Jia17], the content delivery methods presented are classified as Evolutionary and Revolutionary. Evolutionary methods are defined as the ones that collaborate with the traditional content delivery solutions (P2P, CDN, etc.), while the Revolutionary methods collaborate with the emerging content delivery solutions. SDN along with information-centric networks[5], are classified as revolutionary methods because they propose changes in the current protocols and topology.

According to [Jia17] one of the SDN’s advantages is that by centralizing the control layer, the Information Technology department (IT) can configure, manage, secure, and optimize multiple network resources via dynamic automated SDN programs. These programs are written by the IT department directly instead of the multiple vendors of the devices. The advantage of the centralized management is that SDN controllers can manage any switch independent of the vendor and the configuration time of the network devices is greatly reduced and simplified. On the other side, the devices to use must be compatible with the OpenFlow protocol which is the software in charge of installing the forwarding rules (also named flows) on the devices.

### 2.1.1 SDN related works

Once explained the importance of SDNs, in this section we will review some related studies using the SDN architecture. Other useful programs also will be summarized, to understand how SDN capabilities are improving some areas on the networking field.

[5] An information-centric network is a new paradigm to change the internet applications from host-centric end-to-end communication to data-centric communication [Kop07].

First in [Kuz15] Kuzniar et al analyzes the SDN control plane interaction between the controller and real switches and their capacity to obtain and implement the forwarding rules under heavy workload conditions.

In [Wan18] Wang et al uses Mininet (network simulator) to simulate a real network topology with OpenFlow, POX (Python version of the program used as controller), and openVswitch (switch emulator) and distributes inbound traffic to avoid congestions on enterprise networks. This improves the users' Quality of Experience (QoE) and reduces costs in network bandwidth.

For mobile networks Yin et al [Yin19] used SDN to propose a scheme named Hierarchical SDN-based Mobile Management (H-SMM) that uses a hierarchical architecture in the control plane to provide intra-domain and inter-domain mobility simultaneously. This hierarchical architecture proposal is made in response to some issues on SDNs for "being originated in campus and enterprise networks and developed in datacenter networks, in the form of single control domains". Due to the long-distance communication nature in mobile networks Yin et al consider as a necessity the inclusion of both, single and multiple domains. The SDN advantages used at [Yin19] are:

- Increased capacity of the controller to improve the control and programing of the network.
- Improvement of the management provided by a global controller
- Better routing capabilities of local controllers.

Flores Moyano et al [Flo17] propose the application of SDN technology to improve service provision in residential networks. This study focuses on a device installed by the Internet Service Providers (ISP) located at their user's home networks called Residential Gateway (RGW). RGW performs multiple tasks for the network e.g. traffic filtering, remote management, Dynamic Host Configuration Protocol, network address translation, and network address translation. Moyano's proposal is to replace the current RGW which lacks accessibility for the user to perform the required configurations and updates for an SDN-based RGW that uses the programmability introduced by SDNs to insert traffic flows in the data plane devices and increase the flexibility to handle residential network traffic.

Aouadj et al [Aou17] proposes AirNet, a new high-level language for programming SDN platforms. During the research, the authors consider network virtualization as an approach to achieve simplification, modularity, and flexibility of SDN control programs. By creating abstract visions of the physical infrastructure, virtualizations expose only the most relevant information for high-level control policies. Aouadj et al achieves this virtualization by using the edge-fabric abstraction model which separates the network elements in **edges** (support complex network functions and services related to the control plane), **fabrics** (deal with packet transport issues), **data machines** (perform complex operations on packets at the data plane level), and **hosts and networks** (sources and destinations of data flows).

Wen et al in [Wen17] research the detection of errors in SDNs and classify them in two categories. (1) When the error is due to data plane forwarding rules not being active on a switch as expected due to firmware or a hardware glitch. These are classified as missing faults. (2) A priority fault occurs when rules overlap with common matching packets. Since switch control rules can contain wild cards (denoted by *) to match with multiple options on a matching field, SDN assigns a priority value for each flow and SDN process the highest priority rule amongst the matching ones. Both faults (missing faults and priority faults) can lead to undesirable forwarding behavior. Wen et al contemplate that missing faults can be discovered with data-plane probing tools and existing solutions verify rule existence on the switches. But without verifying rule priority order, rule existence cannot warrant forwarding correctness. Thus, RuleScope is introduced and provides a series of inspection algorithms to detect and troubleshoot forwarding faults on the data plane.

To prevent network path failure and handle the increase of data in the network without increasing operational costs by adding more switches and routers, Shamim et al in [Sha18] use openFlow and SDNs with link aggregation control protocol (LACP). LACP allows two or more ports in an Ethernet switch to be combined to operate as a single virtual port. Using link aggregation increases available bandwidth and availability between the devices connected.

In terms of security, Swarmi et al [Swa19] makes a study on SDN security issues and defense mechanisms against Distributed Denial of Service (DDoS) threats. This attack consists in a large (or slow and controlled) volume of packages from multiple sources to be transferred to the network devices, in order to saturate the network’s bandwidth or nodes’ memory and CPU and make the services unavailable for legitimate users. With the decoupling of the data and control planes of the network, SDN switches became simple forwarding devices and considered as dumb. By having a centralized policy, SDNs becomes a convenient target for DDoS. If the control plane breaks down, the complete functionality of the network might be disturbed. Nevertheless, the same features of SDNs (its controller’s global view and programmability) can be used to control the impact of DDoS attacks. In

[Swa19] more detailed information about DDoS, its prevention methods, and consequences are provided.

Finally I will describe the following research by [Dew18] named "Improved load balancing on Software-Defined Network-based equal cost multipath routing in data center network", which is the research in this section that is closer to the goals of this work.

Dewanto et al in [Dew18] increases the dynamicity of equal cost multipath routing (ECMP) by using the controller to monitor the network in real-time for available bandwidths to include in the algorithm and calculate the best path to update the switches. The experiments are performed using Mininet with a RYU controller, a topology that consists of 20 switches and 16 hosts with link's bandwidth of 8 Mbps, and OpenFlow 1.3 protocol. The measured bandwidth is used on the Dijkstra's Widest Path algorithm along with topological information to determine which path has the biggest bottleneck. After the path with the maximum available bandwidth is found, the controller updates the flow tables on the switches. Finally, when the switches finish with the transmission process, they remove the flows previously calculated so the controller can choose a new path again with more recent data of the network. Although Dewanto et al also uses an SDN architecture and Mininet to improve the path selection, the differences with this work lies on the final objective of the research. The results on Dewanto's work are focus on speed improvements and content distribution rather than survivability. Dewanto accomplishes his goal by using a method named Equal Cost Multipath Routing (ECMP) where all possible paths between two nodes are used. We, in comparison to the method used in this work, use an interdisciplinary method to further improve the speed of the search algorithm.

## 2.2 Vehicular and Computer networks

While doing our literature review in some digital libraries (e.g. IEEE Xplore, ACM, ProQuest, Elsevier, University of Winnipeg Library), we found that the idea of combining vehicular networks and computer networks has been previously explored. Several authors have proposed the use of mobile networks, ad hoc networks, or Internet of Things (IoT) to improve information gathering and transit flow (e.g. [Xia15], [Ahn19] and [Muk18]).

According to [Xia15] and [Har08], in vehicular networks there are mainly two kinds of communications, vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I). These communication methods are possible thanks to the low-cost and improvement of global-position systems (GPS) and wireless receiver devices. Some of the objectives for the communication between transit devices are:

- Increase road safety (e.g. avoid collisions, vehicle remote diagnosis)
- Transportation efficiency (e.g. reduce traffic congestion, traffic signal control, route selection efficiency)
- Reduce the impact of transportation on the environment.

An example of the use of internet of things (IoT) on vehicular networks is described at [Ahn19]. With the inclusion of sensors, powerful computing, and communication capabilities, IoT in vehicles to optimize transit processes has become a topic on its own and is called the Internet of Vehicles (IoV). In [Ahn19] Sanghyun Ahn and Jonghwa Choi use the V2I type of communication to estimate the vehicle queue length for traffic signals. Typical methods like video cameras and sensors have deficiencies like high computing overhead, maintenance costs and susceptibility to the environment.

Using the IoV they improve the estimation of vehicle queue provided by typical methods. V2I uses fixed infrastructure RoadSide Units (RSUs) [Xia15], in this case named traffic signal controllers. These controllers reduce the computing overhead and improve the resilience to environmental obstacles.

For the Ad-hoc networks example, in [Muk18] Mukund B. and N. Gomathi use the Lion Algorithm to minimize the routing cost of the VANET. VANETs (or Vehicular Ad-hoc NETworks) are local V2V and V2I wireless networks which offer direct communication with minimal latency. The Lion Algorithm (or Lion Optimization Algorithm) is an optimization algorithm based on lion's social behavior and organization [Yaz16]. Lion Algorithm (LA) provides routing with reduced cost and computational complexity. In their research, Mukund and Gomanthi propose a modified version of the LA. The modification adopts the minimized routing cost under the VANET.

In 2008 Robert Geisberger, Peter Sanders, Dominik Schulter, and Daniel Delling propose the Contraction Hierarchies algorithm. In [Gei08], Geisberger et al propose a method to utilize the structure of road networks to improve path selection. In [Gei08], the authors propose a series of rules to pre-process the network as to identify shortcuts and order the elements of the network. Then the proposed search algorithm uses the pre-processing results to increase the speed of route selection.

Geisberger et al in [Gei12] use the Contraction Hierarchies (CH) algorithm, to pre-process a large road network. The pre-process exploits the inherent hierarchical structure of the network by adding shortcut edges. This pre-process is complemented by another algorithm which calculates the shortest paths (in this case bi-directional Dijkstra). By using the shortcuts produced by the pre-process, the path calculation algorithm can optimize the search. The motivation was to create a simple algorithm which can be adapted to multiple situations. The efficiency and simplicity can be used by mobile devices to search for fast routes with a short delay for the user.

Another relevant research we found for our purposes focused on the vehicular networks area is found in [Xia15]. In [Xia15] Xiao and Kui use the data gathered by a taxi company to trace a roads map with the frequent traces of taxi locations. Then they use the map and SDN architecture to improve vehicular path selection by connecting fixed infrastructure Roadside Units (RSUs) of the VANET to the SDN controller. The inclusion of the SDN paradigm provides the system with important innovation characteristics, e.g. resource optimization, packet routing and forwarding, and efficient mobility management. The addition of routing and forwarding produced by the centralized controller is especially important to greatly promote the efficiency of the vehicular system.

Although Xiao and Kui in [Xia15] focused more on vehicular networks, the importance of this paper is that it makes clear that SDNs and vehicular networks are alike due to certain similar characteristics on both systems. The result of the research in [Xia15] is a map similar to the infrastructure of road networks in which SDN is applied. Thereby, we believe that (with few modifications), the algorithms used for road networks like the Contraction Hierarchies algorithm can be applied to optimize computer networks.

## 2.3 Importance of Mininet

While doing research on networks, one of the main problems faced before doing experiments is to select a suitable, affordable, and realistic testbed that can be configured to the research needs. In this section we will analyze other options available to carry out network experiments from the point of view of other authors. We also review some of the pros and cons found while selecting the software used for this research (i.e. Mininet software).

In [Lan15] and [Lan10] Lantz et al highlights the problematic that researchers face while using heavyweight and full machine virtualization programs, as well as the option of paying for an expensive testbed. This is an important consideration due to the complexity and overhead cost of simulating and manage large networks using a virtual machine for each node in the network, which can end in a reduction of realism, usability, and scalability of the development platform.

Other research platforms are more realistic by using real servers in different locations. But instead of paying for the services, they might ask researchers to add resources to the network by contributing with the infrastructure of these platforms, or to contribute in a different way other than money. These option helps the network to grow larger, but sometimes is less affordable than money due to the nature of the resources, the requirements for the experiment, and the time that the experiments will last.

Mininet can create a large network in a single laptop with Linux features using a single and simple python API. The network can contain gigabits of bandwidth and hundreds of nodes (switches, hosts, and controllers). The entire network can be packed as a virtual machine so others can download, examine, and modify it. These characteristics help the user to avoid installation of unnecessary software and complicated management.

While Mininet provides an easy to manage, escalate, and lightweight testbed for networks and SDNs, it is by no means perfect. Jiaqui Yan and Dong Jin [Yan17] introduce the notion of virtual time for containers to improve Linux based technology for SDNs. This work's premise is that Mininet lacks in terms of fidelity because the operative system serializes the virtual machines rather than the parallel scenario of a physical testbed. The serialization and the limitation of physical resources can limit the scalability capabilities of the system.

Nevertheless, this research's goal is to improve the characteristics of SDN routing by taking advantage of the similarities between road and computer networks. The similarities will allow us to adapt the algorithms from road network to work on computer networks. As the improvements are evaluated while we simulate errors in the network, we highlight the value of a re-routing extra module in the SDN algorithm. This extra module must have the capacity to adapt and improve the content distribution while answering to unexpected errors in the network. To fulfill this purpose, the capabilities of Mininet are considered sufficient for the realization of our experiments. All we require from Mininet is a script to randomly stop the switches while the transmissions are in process, to prove the efficiency of our algorithm for solving these issues. In addition, the algorithms will be executed in the RYU controller. The controller is linked to Mininet but it does not depend on Mininet's execution, and the serialization can be (to a certain degree) controlled by managing the threads with code. Thus, we believe that the serialization of the processes does not greatly affect the results of the experiments.

# Chapter 3

## SDN Environment

In this chapter we present the general setup to be used for the algorithms testing, including the programs selected, the topologies and some backup programs that will improve the realism of the experiment's results. First, in Section 3.1, the general architecture of the experimental environment is defined and described. In this section we will explain how all the programs will interact with each other to produce the desired results. In Section 3.2, we describe the programs and protocols used on the experiments. Finally, in Section 3.3, we introduce the topologies used and the reasons why these where selected.

## 3.1 SDN General Architecture

In this section we describe the general architecture of the network used for testing. The general architecture can be seen in Figure 6. In this section we describe the relation between the different elements of the environment.

The entire SDN environment is simulated in a virtual machine with Ubuntu 18.04. Mininet, as seen in Figure 6, is used to implement the entire data plane. The options introduced while executing Mininet allows us to run other functions of the data plane, such as:

- A script on Mininet is used to simulate the custom virtual infrastructure (Switches and Hosts of "custom topology" in Figure 6).

- The traffic on the data plane is simulated by the Distributed-Internet Traffic Generator (D-ITG) introduced in Section 3.2.5
- Finally, for the data plane, Mininet uses OpenVSwitch to simulate the software of the switches.

Complementary to Mininet (data plane), the controller (control plane in Figure 6) is run using the RYU controller. Even though the network (hardware and software) is entirely simulated on Mininet, the code used by the controller to manage the traffic of the network can be used on a real controller setup.

RYU controller contains several modules with files that manage the behavior of the virtual switches. The network administrator can select and modify the specific file that contains the code to be implemented. As seen in our SDN environment (Figure 6), in the controller we include the different functions, algorithms, sequence and analysis that the program will follow when receiving flow requests from the switches. Our code is used to plan and balance the routes for the traffic in the network. In addition to the analysis program, we can add multiple modules used to discover and manage the network elements. Once the network elements are identified, we can also include code to obtain information in real time to use in the flows.

Finally, as seen in Figure 6, to connect both the control and data planes, we use OpenFlow. This allows the controller to obtain the information of the network elements and allows Mininet to do queries for the routes and receive the flows to redirect the information.

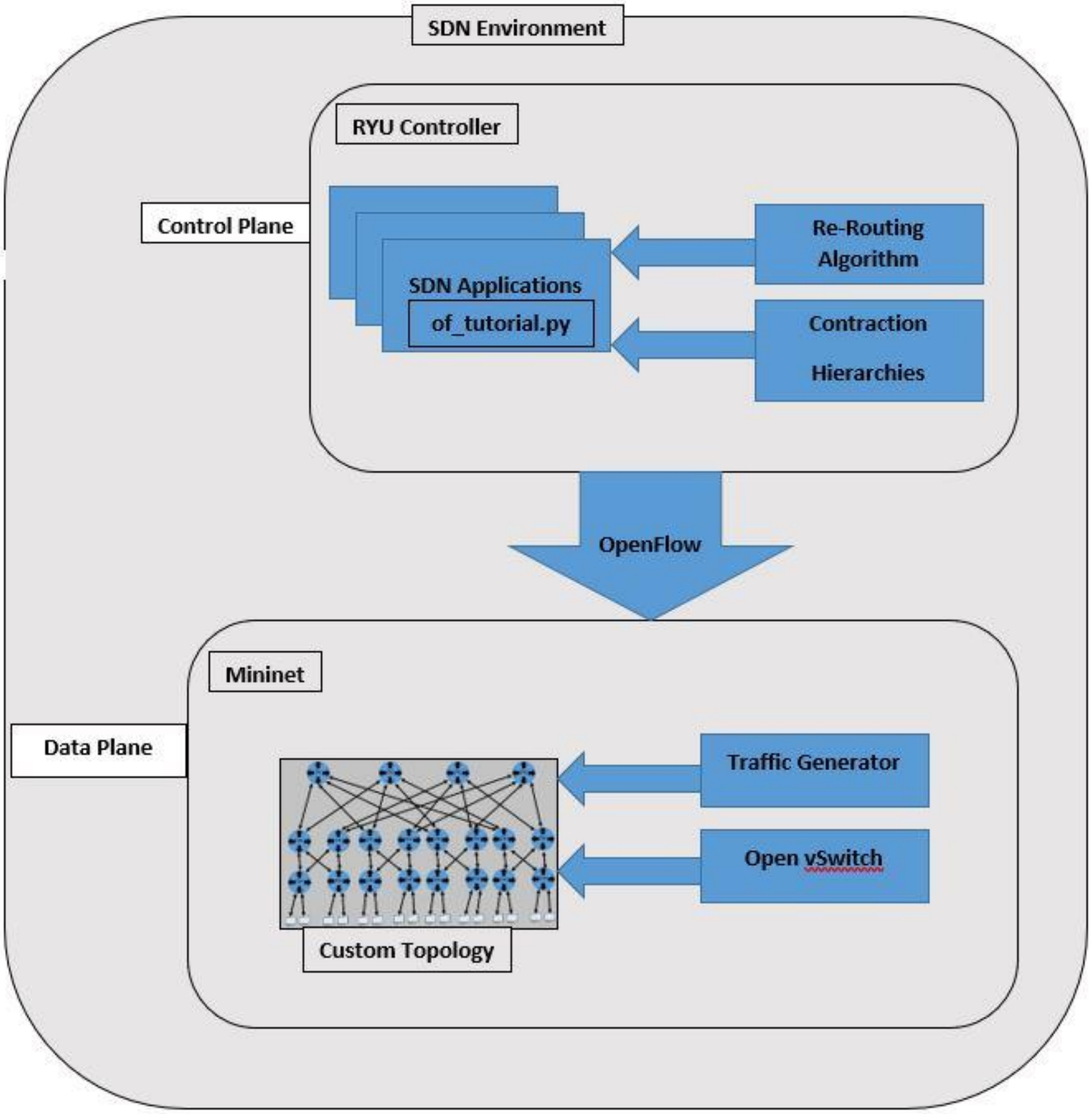


*Figure 6 SDN General Environment*

## 3.2 Programs and protocols used

### 3.2.1 Software-Defined Networks

According to the Open Networking Foundation [Onf19], Software-Defined Networking is the physical separation of the network control plane from the forwarding (data) plane. The control plane in its basic form is delegated to a device named controller which dictates the logic of how the forwarding plane will work. The controller can coordinate several devices at the same time, as long as the devices are compatible with the protocol in charge of communicating with the SDN controller (e.g. OpenFlow). More than one controller can be used on the architecture. These multiple controllers can improve the network control plane's scalability, availability, and prevent single points of failure.

By separating and centralizing the control plane, administrators can manage the entire SDN-based infrastructure from the SDN controller(s). This centralization allows administrators to dynamically adjust network traffic according to their needs or according to the applications on the network. These changes can be programmed by the administrators on automated SDN programs due to the neutral nature of the OpenFlow protocol, which allows the communication with the network elements independently of their vendors. The improvement in programmability makes possible the faster inclusion of network infrastructure, which can use the same programs on the controller to produce the flow tables it needs to work.

In the SDN architecture, after the control plane is taken to the controller, the devices of the data plane are left as "dumb" devices who's only function is to forward the packages. When a package arrives to the data plane device, it is sent to the controller to be analyzed and to decide the path to take. The paths to follow are received by the network devices as flows. These flows are the rules to which the next packages will be compared to forward the packages.

To determine how the packages will be forwarded, the controller connects on an upper layer named application layer. This application layer (also known as application plane), contain SDN applications executed on the controller that control the network's behavior. Figure 7 shows the logical architecture of the SDN networks, which combines the data, control and application planes with the northbound APIs and southbound APIs[6].

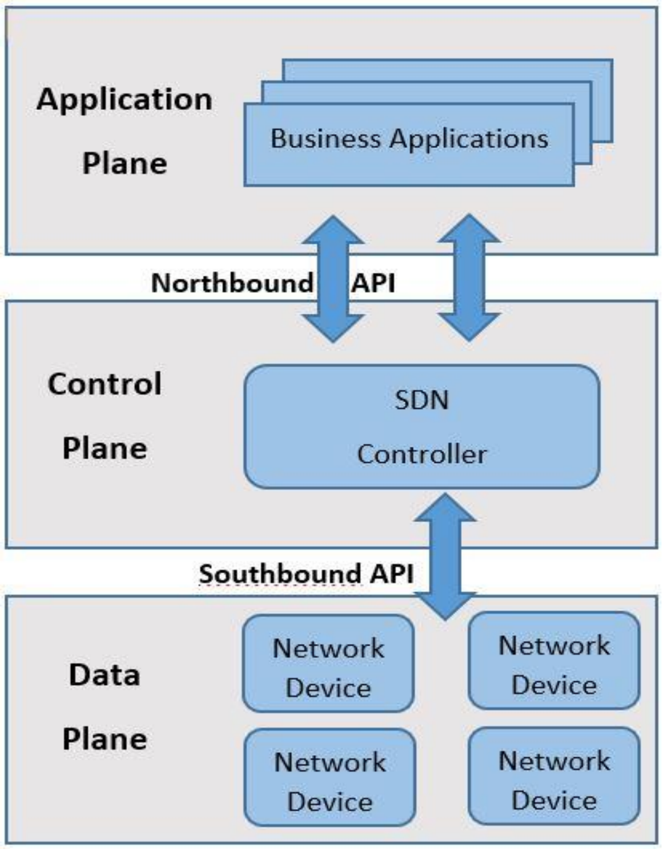
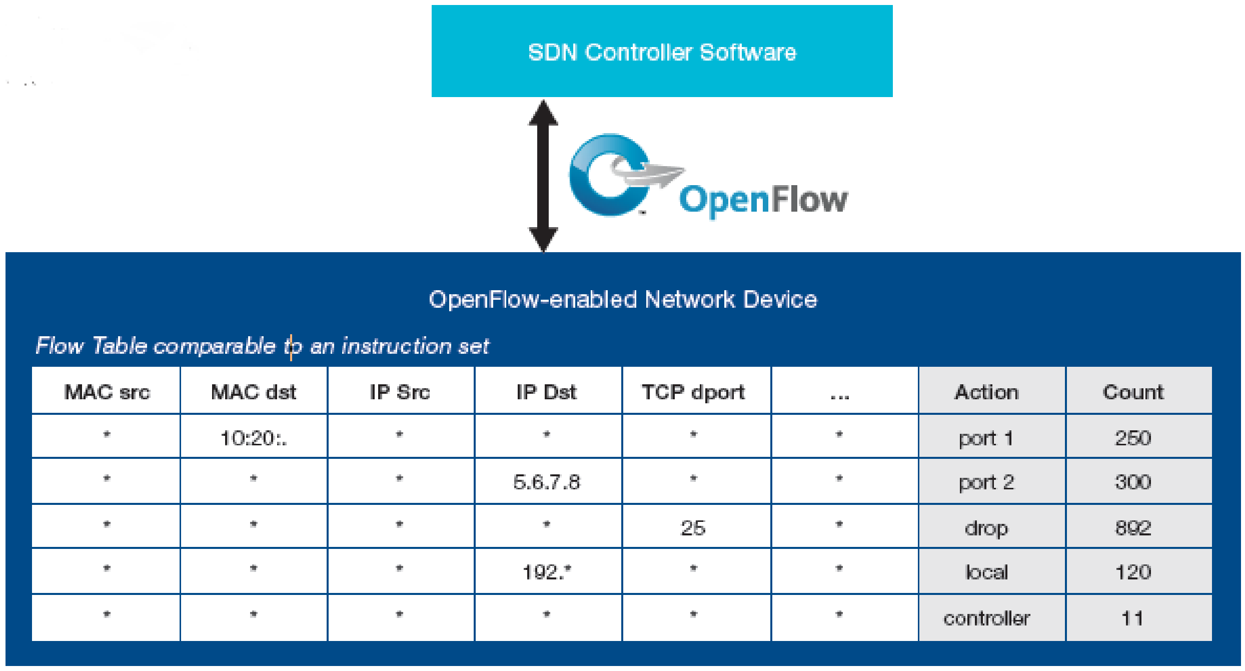


*Figure 7 SDN Architecture*

[6] OpenFlow is the standard southbound API to connect the control and data planes.

### 3.2.2 OpenFlow

OpenFlow is the first and most widespread open SDN standard protocol to connect the control and the data planes [Mah18]. It can be described as a forwarding table management protocol where a group of forwarding tables are maintained in each of the OpenFlow forwarding elements (switches, routers, etc.). The forwarding table consist of forwarding/matching rules called flow rules, which dictate the operation performed for each packet that matches a flow rule upon arrival.


SDN Controller Software
OpenFlow
OpenFlow-enabled Network Device
Flow Table comparable to an instruction set

| MAC src | MAC dst | IP Src | IP Dst | TCP dport | ... | Action | Count |
|---|---|---|---|---|---|---|---|
| * | 10:20:. | * | * | * | * | port 1 | 250 |
| * | * | * | 5.6.7.8 | * | * | port 2 | 300 |
| * | * | * | * | 25 | * | drop | 892 |
| * | * | * | 192.* | * | * | local | 120 |
| * | * | * | * | * | * | controller | 11 |

*Figure 8 Flow table. Example obtained from [Onf12]*

Figure 8 shows a representation of a Flow Table in SDN networks found in [Onf12]. As seen in Figure 8, some of the characteristics on the flows are:

- In the case a package characteristics match with the characteristics defined on the flows, the package will perform the action defined by the matching flow (row in the table).
- The flows can contain a default field defined by *, this means that only the defined fields will be compared.
- The flow at the bottom which contains only default fields (*), will send the packages to the controller to be analyzed and install new flows if needed.

Additional to the fields on Figure 8, the flow tables contain a column named priority. This priority field is a numerical value that determines the order in which the flows will be executed in the following way:

- If a package matches with more than one flow, the flow to take action will be defined by a priority field defined on the flows.
- The flow with the highest priority will be the action to take by the packages.

  The importance of a proper priority number selection is that, if a flow like the one who sends the packages to the controller takes a higher priority, all the packages will be sent to the controller, even if the route has been previously defined. Furthermore, the route selection can be improved or distorted by the flow selected by the priorities.

In the document "Software-Defined Networking: The new norm for networks" [Onf12] from the Open Networking Foundation the authors define the OpenFlow protocol as the basic primitives that can be used by an external software application to program the forwarding plane of the network devices.

This protocol must be implemented on both, the software of the SDN controller and the infrastructure devices that will apply it.

SDN architecture provides multiple benefits to the network [Onf12]:

- Centralized control of multi-vendor environments: as long as the network element supports the OpenFlow protocol, the controller, regardless of their vendors, can manage the elements.
- Reduced complexity through automation: SDN management framework makes it possible to develop tools that automate management tasks.
- Higher rate of innovation: SDN allows network operators to program and reprogram the network in real time to meet specific business needs and user requirements as they arise.
- Increase network reliability and security: SDN allows the network operators to define high-level configuration and policy statements.
- OpenFlow-based architecture eliminates the need to individually configure network devices which reduces the likelihood of network failures due to configuration or policy inconsistencies.
- OpenFlow control model allows network operators to apply policies at a granular level.
- SDN infrastructure can better adapt to dynamic user needs.

Although SDNs still have some problems, identified by Wen et al [Wen17] and Swarmi et al [Swa19] including:

- Missing fault occurs when a rule is not active on a switch mainly attributed to the switch's firmware or hardware or a rule-update message lost.
- Priority fault occurs when two or more overlapping rules (with common matching packets) violate the designated priority order, since SDN packages are processed by the highest-priority rule amongst the matching ones, this can lead to undesirable forwarding behavior.
- Single point of failure, in SDN cases produced by the controller(s) which centralizes the control plane of the network.
- Security attacks to the network elements, especially to the controller which can greatly obstruct the network performance.

### 3.2.3 Mininet

This brief explanation of Mininet's importance is a complement of Chapter 2 Section 2.3. This section focuses on information of how Mininet will be involved to the main infrastructure, instead of the related works of the pros and cons of using Mininet.

One of the most important phases of research is testing. During this process, we test the ideas defined into the hypothesis to obtain some results to support our research. The environment of the tests plays an important role in the veracity of the results. As discussed in Chapter 2, network testing needs to contemplate fault tolerance and scalability. Full machine virtualization and heavyweight containers increase the network complexity, increase the overhead, and reduce the usability and scalability [Lan15]. Also, heavyweight containers often consume significant hardware resources and requires complicated management.

Mininet is a system for rapidly prototyping large networks on the constrained resources of a single machine. Using OS-level virtualization features (processes and network namespaces), it allows the network to scale to hundreds of nodes. Using Mininet, the user can dynamically create customized virtual testbeds. Mininet supports lightweight virtualization and allows users to implement a new feature or architecture and test it on large topologies with application traffic. The entire network can be packed as a VM for others to download it, run it, examine it, and modify it. The use of Mininet is important for this work because:

- Allows the connectivity with SDN controllers and virtual SDN switches.
- Easy to program and change personalized network topologies.
- Allows the use of multiple and customized controllers.
- Includes a command line to run useful commands and scripts with multiple commands.
- Allows the use of additional xTerminals for additional and individual command execution on the network elements.
- Easy management and inspection of virtual switches.
- Open software easy to install and configure.

These characteristics allow us to use Mininet as the core program to simulate the data plane of the network. The easy connectivity with the controller allows

Mininet to obtain the flows needed to distribute the packages and observe the changes through xTerminals. XTerminals are individual windows that can be setup for each node in the Mininet infrastructure to run more complex and interactive commands that require constant or parallel execution. The xTerminals also allows the connection with other programs to improve the veracity of the network. Finally, each xTerminal allows the individual management of the nodes, to improve the control over the network.

### 3.2.4 RYU Controller

A controller in the SDN paradigm is an element in the network that centralizes the control plane of the forwarding devices in the network on a component that supplies flow table rules to OpenFlow devices. This element manages, centralizes and makes programmable the package forwarding process to improve the data layer in the network.

The selected controller for this work was RYU controller due to the characteristic of the code which makes it easy to understand, easy to manage, and the functions needed for the experiments worked as intended[7]. For more information, from the RYU getting started homepage we can highlight the following characteristics [Ntt11]:

“RYU is a component-based software-Defined Networking framework.

RYU provides software components with well-defined API's that make it easy for developers to create new network management and control applications. RYU supports various protocols for managing network devices, such as

[7] Other controllers were tested during this research, but the results and execution did not work as expected of the program.

OpenFlow, Netconf, OF-config, etc. About OpenFlow, RYU supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions.

All the code is freely available under the Apache 2.0 license. RYU is fully written in Python."

Other alternatives instead of a RYU controller are listed next [Git19]:

- Beacon (Java)
- Floodlight (Java)
- Trema (Ruby)
- POX (Python)
- NOX

Our reasons to select RYU were:

- The programing language Python, which is easy to understand, and we have experience working with it.
- The complexity of the code which uses few functions and validations but keep the code simple and understandable for the user to program personalized functions.

### 3.2.5 D-ITG (Distributed Internet Traffic Generator)

Multiple types of systems, who generate different types of objects and data, affect networks nowadays. To test and develop these new systems, an important factor is the generation of network workload that represents the variety of network data. There can be two types of traffic generation, by hardware or by software. According to Botta et al in [Bot12] a realistic network workload approach should:

- Realistically represent the complexity of a real, specific network scenario(s).
- Measure indicators of the performance experienced by such workload.
- Allow the alteration of specific properties of such workload for the purpose of the experiment.

As shown in the D-ITG manual by Botta et al. [Bot13] D-ITG is a platform capable of producing and analyzing IPv4 and IPv6 traffic. D-ITG uses the most common performance metrics (e.g. throughput, delay, jitter, packet loss) at packet level. At the transport layer, D-ITG supports several protocols e.g. TCP, UDP, SCTP, DCCP, and ICMP. For the final point to consider for traffic generators we have the characteristics of ITGSend and ITGRecv. ITGSend is the sender component of D-ITG and work in three different modes:

- **Single-flow** read the configuration of a single traffic flow toward a single ITGRecv (D-ITG receiver module).
- **Multi-flow** read the configuration of multiple traffic flows toward one or more receivers through a script file.
- **Daemon** run as a daemon listening on a UDP socket for instructions.

Finally, D-ITG is a software type of traffic generator. [Unk15] shows that D-ITG is compatible with the Mininet configuration proposed in this work.

## 3.3 Topologies for path selection

### 3.3.1 Fat Tree topology

Fat Tree is a variation of the regular Tree topology where the special characteristic is that the number of nodes going to higher layers is the same number of nodes going to lower layers. As a variation of the tree topology (see Figure 9a), fat tree uses a hierarchy relationship between the layers with the core at the top (see Figure 9b). The core contains less switches than lower layers. Nevertheless, the higher the layer, the capacity of the switches also increases, so the higher layers are said to be thicker than the lower ones. On the bottom layer connecting the servers or hosts, we have the edge layer. Below the core layer we have the aggregation layer which distributes the information between the edge and core layers.

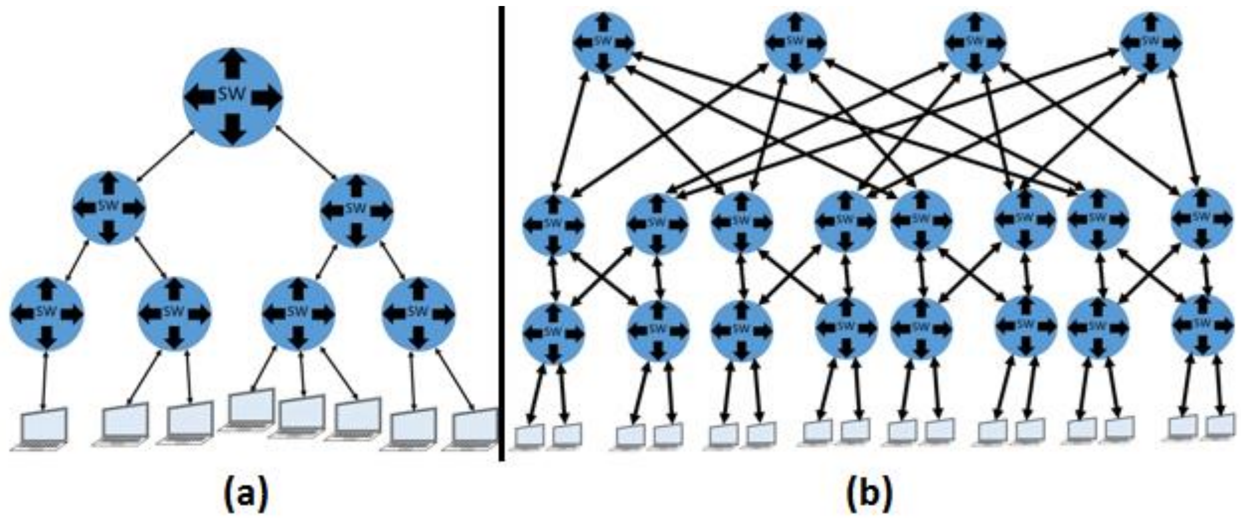


*Figure 9 (a) Tree Topology  (b) Fat Tree Topology*

One of the characteristics of the fat tree topology to be used on this work is the hierarchy of the switches on the fat tree topology. Also, the multiple paths to reach the different hosts. In terms of path selection and content distribution, having a second path available through different switches to reach the same destination can produce loops. Nevertheless, depending on the algorithm selected for the routing, the multiple paths should not be a problem and can even be an advantage, in our case if a switch gets disconnected and we need to choose a different path. Under normal circumstances. The bi-directional Breadth-First Search algorithm keeps track of the visited nodes (for more information, see Chapter 4 pseudo algorithms). The visited node list allows the algorithm to evaluate if a route search is returning to a previous node to avoid loops in the search. While sending the information, the switches use the flows obtained from the controller to decide where to redirect the packages.

### 3.3.2 Custom topology

Other than the fat tree topology, we needed a smaller topology which allow us to have multiple case scenarios but in simpler forms than those of the fat tree. While using this topology, we were able to variate it according to our experiment needs which improve the analysis and gave us some ideas to perform our algorithm. The final form of this topology can be seen in Figure 10, nevertheless multiple switches were added or deleted as needed to perform our tests.

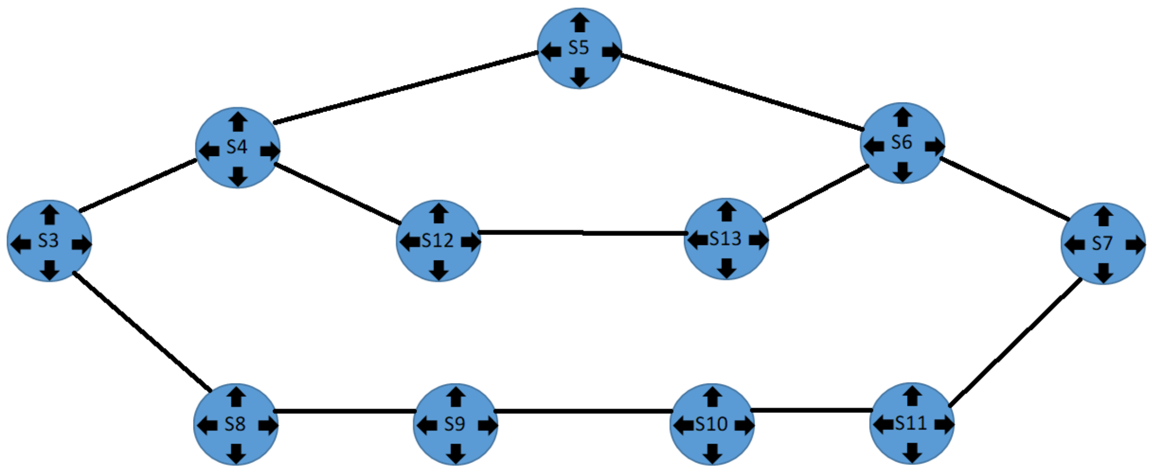


*Figure 10 Custom Test Topology*

# Chapter 4

# Path Selection Implementation

In this section, we introduce the pseudo-algorithms to use for the experiments in this research. The first algorithm to implement is the pre-processing phase of the Contraction Hierarchies Algorithm (CH). CH is originally used and proposed for road networks. The second algorithm corresponds to the query phase of our work. We implement a modified bi-directional search which uses the advantages obtained from the Contraction Hierarchies algorithm. The third algorithm responds to unforeseen events on the network. We implemented a recovery algorithm in cases of switches failing and use the advantages of the CH and SDNs to optimize this process.

## 4.1 Pre-Processing Phase

### 4.1.1 Contraction Hierarchies and Road Networks

Road networks is an area of traffic management that, as computer networks, is in constant study and optimization due to its everyday importance.

Contraction Hierarchies CH is an algorithm proposed for road networks by Geisberger et al. in [Gei08] based on the hierarchical nature of road networks. Although the authors of the CHs algorithm focused on using CH for road networks, it can be implemented on graphs that hold similar characteristics to the hierarchical abstraction of a road network. The computer networks characteristics and the inclusion of SDN allow us to implement CH (with few modifications) on computer networks.

The abstraction of road networks consists of roads with certain cars capacity and direction. These roads can be small streets that go only one-way, two-way streets, avenues, and highways with high speed and high capacity, etc. To arrive to their destination, drivers must take multiple roads, and the way to change between roads is to take a turn on intersections. A general trip often goes from a small slow street to another road that allows the user to travel faster (and so on) in order to decrease the travel time, then back to another slower road which finally will connect to a destination. While we mention slow or fast travel, this not always refers to speed. In our everyday lives, multiple factors affect the way we travel and according to our needs we might need to optimize different parameters. This gives road networks a simple hierarchy according to road capabilities and frequency of use. Finally, if a driver traces a path from the streets connecting its source and destination points to a higher hierarchical road, there will be a point (or multiple points) where these paths can intersect. By calculating the shortest intersection of these routes, we can obtain the shortest path between a defined source and a defined destination. By making the analogy with computer networks, we can say the following:

- The roads, car flow direction, and car capacity work as the connection between network elements and their bandwidth. We can classify these network connections into hierarchies depending on the parameters to calculate the fastest routes.

- The intersections distribute the traffic through different paths as network elements distribute the packages through the links.
- Hosts work like the starting point and destination of the drivers.
- The drivers use roads in similar ways to the packages of the computer networks.
- Finally, to trace the routes while considering the variables affecting the traffic, drivers can use devices who have a broad view of the network (like GPSs). With the implementation of SDNs, we can obtain similar information to improve our network routes.

These similarities can be applied in different ways by researchers. In previous examples (Chapter 2 Section 2.2), road infrastructure which is located at important points of the roads (like intersections) are the ones connected to the SDN infrastructure (controller). For this example, the network infrastructure is the one acting like the roads and intersections. The direction of the roads is discarded because in our implementation the links between network devices work both ways.

The CH algorithm is a path optimization algorithm for road networks, which uses the pre-processing phase to improve the path selection phase. As the network becomes larger, the pre-processing phase times increase but allows an improvement compared to normal search and bi-directional search algorithms. A good performance on the pre-processing phase can lead to a greater optimization on the routing phase. In addition, the re-routing phase added in this work can also be enhanced by simplifying the new routes by using SDNs and the CH results.

Before explaining the Contraction Hierarchies algorithm, the following points are important to understand the pseudocode. As explained before, we can use the formulas from the original algorithm [Gei08] with few modifications.

Considering a graph G with vertices V (nodes) from 1-to-n and edges E (links) from 1-to-m

$$G = (V, E)$$

- We consider a node $v \in V$ contracted when we temporary remove it from the graph in order to replace paths of the form $[u, v, w] \in V$ by the shortcut $[u, w]$.
- During contractions, u and w are nodes from V with higher priority than v.
- To determine if a shortcut is needed, we compare other paths $[u, w]$ that does not go through v. A shortcut is added if the shortcut [u, w] is the shortest path from u to w.
- To calculate the priority of a node v we use the ***Edge Difference (ED)*** method and the following equation.

  $$ed(v) = s(v) - in(v) - out(v) \qquad \text{(Equation 4.1)}$$

  ED is the difference between the number of links contracted and the number of shortcuts that can be created.

  - S(v): number of shortcuts when contracting v
  - In(v): links with destination v
  - Out(v): links with source v

- After contracting a node v, the priority of the neighbors can change. The new priority values of the neighbors are recalculated, and the order can be modified.

### 4.1.2 CH pre-processing pseudo algorithm

In this algorithm we have the information of the network as input, and the shortcuts and node order as outputs. “Adjacent_nodes” is an important function in which, given a specific switch id, returns the links and shortcuts where the source is the input switch.

First, we calculate the importance or edge difference of each node v. To do this, we use the “calculate_edge_difference” (ED) function, this function uses the “adjacent_nodes” function to obtain the second and third value of the edge difference equation[8] 4.1. To obtain the first value, we use the third procedure “alternative_routes”. While searching for alternative routes, given two adjacent nodes to v, we search for a path between them other than through v. This procedure can be limited to a search for a route different than v with “max_len”[9] jumps. After calculating all the “ed” values of all the nodes, we select the node with the highest “ed” value and use the procedure “node_contraction” to contract the selected node. The result of contracting a node is to “eliminate” this node temporarily and to add the necessary shortcuts. The node contracted receives a contraction number based on the turn it was contracted. Finally, we re-use the “adjacent_nodes” and

[8] As mentioned before, the direction of the links is irrelevant, so input and output links are reduced to the same “links” value.

[9] Max_len is the number of jumps from v to the adjacent nodes being evaluated src-v-dst contains 2 jumps. The number can increase when the adjacent node is a shortcut.

“calculate_edge_difference” procedures to re-calculate the “ed” value of the neighbor nodes and repeat the process until all nodes have been contracted.

**INPUT:**

```
Links_Information
    //Dictionary with port objects
    //(Port number, switch ID, active flag)
    //A link is a pair of ports = {source_port, destination_port}

Active_switch
    //Dictionary with active switch information
    //(ID to match with links)
```

**OUTPUT RESULT:**

```
Shortcuts
    //of the initial topology

Node_order
    //order of importance
```

**PROCEDURE:** adjacent_nodes[sw]
//Uses link and shortcut information
//return adjacent nodes

```
For  link in links_information do
    if link.source.id equals sw.id then      //compares link source
                                             //and function input switch
        adjacent ← link
    End
End


For shortcut in shortcuts do
    if shortcut.source.id equals sw.id then
        adjacent ← shortcut
    End
End

Return adjacent                    //adjacent links/shortcuts to
                                   //input switch (sw)
```

**PROCEDURE:** Calculate_edge_difference[]
//inspired on the equation 4.1

```
For switch(sw) in active_switch do
                    // Calculates initial importance of the switches
    adjacent_nodes[sw]
    $ed(v) = s(v) - in(v) - out(v)$
                    //Importance = possible shortcuts – In/Out links
    Importance[sw] ← ed
End                 //The "ed" value defines the order to contract the nodes
Return Importance
```

**PROCEDURE:** alternative_routes[start, stop, v, max_len]
//simple search of paths different than v

```
Level = 1
Visited = {}            //nodes visited = {node_route: level}
Visited = {start : 0} //nodes visited and jumps from node "start"
For i = 0, 1, 2, … max_len do
    For node in visited do
        If node.level equals level - 1 then
            Adjacent = adjacent_nodes[node]
            For adjacent in Adjacent do
                If adjacent equals stop then
                    Return true
                Else
                    Visited ← {route_to_adjacent: level}
                End
            End
        End
        Level = level + 1
    End
End
Return false
```

**PROCEDURE:** node_contraction[sw]
//Contracts the higher importance node

```
Adjacent = adjacent_nodes[sw]

Node_order ← {sw: order_number} //order number increases by 1
                                //each contraction

For x in range(len(adjacent)) do      //len is the int number of
                                      //elements in adjacent

    For y in range(x + 1, len(adjacent)) do
                                      //range(start, stop, increase(def = 1))

        Max_length = len(adjacent[x]) + len(adjacent[y])

        Alternative_routes(adjacent[x], adjacent[y], sw, max_length)
                                      //alternative_routes returns T or F

        If alternative_routes equals false then

            shortcuts ← adjacent[x] + sw + adjacent[y]
                                      //shortcut = src-v-dest

        End

    End

End
```

## 4.2 Query (Search) Phase

After performing the pre-processing phase of the Contraction Hierarchies algorithm, we must select an algorithm to manage the queries for routes. This algorithm will calculate the shortest path from the Source to Destination (S to D) using the shortcuts of the CH algorithm.

We can find multiple options in literature depending on the type of network we are managing and the conditions of the source and destination. For trees the simplest option is Depth-First Search (DFS). DFS has a big problem, in our examples the source changes so the top of the graph is different on each query. While it is possible to rearrange the graph to solve this problem, other available options are better for this case scenario.

A common practice on graphs is to assign a weight value to each path between the nodes, then we calculate the value of the distance between source and destination based on these weights.

One of the most used algorithms used to find the shortest route from S to D is the Dijkstra algorithm. The Dijkstra algorithm (or Dijkstra's shortest path first algorithm) is an easy to implement and easy to understand path algorithm. This algorithm uses the weights of adjacent nodes to the source to select the path to follow. Then it selects the path to the adjacent node with the minimum weight and adds the weight of the following adjacent paths. The process of adding the weights of the routes and selecting the minimum path value continues until the destination node is reached. This process guarantees that the shortest path in terms of the weights is selected.

In case we don't have the values of weights or these values were not previously assigned to the graph, we can use the Breadth-First Search (BFS) algorithm. This algorithm works in a similar way to the Dijkstra's algorithm but instead of the weights, BFS uses the number of jumps as the metric to determine the shortest path. One of the advantages of BFS is that it keeps track of the visited nodes, this allows the algorithm to work even if loops are present in the graph. BFS as well as Dijkstra, continues until the destination node has been found and returns the path as well as the distance (number of jumps) of the shortest path.

Finally, one of the main optimizations to the Dijkstra and BFS algorithm is the bi-directional search (BS). BS can be implemented when both the source and destination of a search are static. The BS process uses the same search methodology from Dijkstra and BFS but instead of searching from S to D, it searches both ways at the same time (S to D and D to S). The process ends when both graphs of visited nodes intersect, and the minimum path can be calculated. BS reduces the complexity of the path search.

### 4.2.1 Changes to the algorithm during the searching phase

Due to the multiple parameters that can affect a network's performance and the lack of a real case scenario with real performance data, in this research we follow a general approach on the searching parameters. To take advantage of the SDN paradigm and CH algorithm we implement the search algorithm proposed in [Gei08] with a unidirectional jump-based approach. This means that the algorithm used is a bi-directional BFS on an unweighted graph. The benefits from CH remain the same based on the number of jumps, this means that the BFS algorithm is both, limited by the node order and improved by the shortcuts obtained in the previous phase.

### 4.2.2 Query pseudo algorithm

For the convenience of the query algorithm we create a different function in which the results of the pre-processing phase are used. The previously introduced (in Section 4.1.2) “adjacent_node” procedure is re-used in this section, but some modifications are implemented. While searching for adjacent nodes, we limit the node search to those which have a destination with a higher node contraction order (or node order) value than the source. All the searches for nodes in the Query section are performed with this modified “adjacent_node” function.

To calculate the shortest route, first we search for adjacent nodes to the source and destination nodes and place them on level 1. Then we search for adjacent nodes to those on level 1 and place them on level 2 and so on. This process is repeated until both searches return cero adjacent nodes and a shortest route can be calculated. After calculating the route, we use a global variable to keep track of the active routes and we implement the route on the necessary switches. The final process is implemented on the third procedure “Install_flow_route” which install the flows on all the switches of the required route in both directions when an answer is needed. The flows can be specific to a protocol so the administrator can change the configuration to install the flows from source-to-destination and destination-to-source in one go, or only from source-to-destination.

**INPUT:**

```
Source(src)
    //src is an integer variable that represents the ID of the source
    //switch obtained from the package that triggers the query event
Destination(dst)
    //dst is an integer variable that represents the ID of the destination
    //switch obtained from the active_switch variable in Section 4.1.2
    //and the dst mac address on the package that triggers the event
```

```
Node_order
    //int list with the order in which the nodes were contracted
    //obtained as output from the pre-processing phase (Section
    //4.1.2)

Shortcuts
    //String list with the shortcuts of the network
    //obtained as output from the pre-processing phase (Section
    //4.1.2)

Active_switch
    //same as the input from last algorithm (Section 4.1.2)

Links_information
    //same as the input from last algorithm (Section 4.1.2)
```

**OUTPUT RESULT:**

```
Global_routes
    //string array with the calculated routes as Src-V-Dst

Flow_route
    //install flows in switches from the previous route
    //not an actual variable
```

**PROCEDURE:** adjacent_nodes[sw]
//Uses link and shortcut information, return adjacent nodes

```
For link in links_information do
    if link.src.id equals sw.id and link.dst.order > sw.order then
            //compares link source id with function input switch id and
            //destination order must be a higher order than source order
        adjacent ← link
    End
End

For shortcut in shortcuts do
    if shortcut.src.id equals sw.id and shortcut.dst.order > sw.order then
            //compares link source id with function input switch id and
            //destination order must be a higher order than source order
        adjacent ← shortcut
    End
End

Return adjacent
        //adjacent links/shortcuts to input switch (sw)
```

**PROCEDURE:** bi_directional_breadth_first_algorithm[src, dst]

```
Level = 1

Visited = {}                    //nodes visited = {node_route: level}

Visited = {src: 0, dst: 0}      //route to nodes visited and
                                //jumps from node “start”

If route[src_dst] is in Global_routes then
    Return route[src_dst]
End

For node in visited do
    If node.level equals level - 1 then
        Adjacent = adjacent_nodes[node]
        For adjacent in Adjacent do
            If adjacent.order_number < node.order_number then
                Ignore lower order nodes
            Else if adjacent in visited and source_of visited.node_route
            not equal to source_of adjacent.node_route
                Global_routes ← Calculate_final_route()
                Return Calculate_final_route()
            Else
                Visited ← {route_to_adjacent: level}
            End
        End
    End
    Level = level + 1
End
```

**PROCEDURE:** install_flow_route[route, src_ip, dst_ip, package]

```
priority = len(route)

package_type = package.ether_types            //protocol of the package
                                              //received by the controller

For switch in range(len(route) – 1, -1, -1) do  //reverse loop through the route

    If switch is the last element in route then

        Out_port = port_connecting_to_dst_ip

        In_port = Link_information.port_number where src = switch
                and dst = route_previous_element

    If switch is the first element in route then

        In_port = port_connecting_to_src_ip

        Out_port = Link_information.port_number where src = switch
                and dst = route_next_element

    else

        In_port = Link_information.port_number where src = switch
                and dst = route_previous_element

        Out_port = Link_information.port_number where src = switch
                and dst = route_next_element

    End

    Datapath = Active_switch.switch.datapath

    Add_flow(switch = datapath, priority, package_type, source = src_ip,
            destination = dst_ip, actionOutput = out_port)

    Add_flow(switch = datapath, priority, package_type, source = dst_ip,
            destination = src_ip, actionOutput = in_port)

End

Datapath.send_msg(datapath, actionOutput = out_port, package)
```

## 4.3 Recovery Procedure

For the recovery phase we use the built-in functions of the controller to trigger an event when a switch is disconnected. At this point we can execute the functions we need to recover the paths. The recovery process consists in two phases, the first phase tries to recover the active routes, and the second phase evaluates when we need to re-do the complete route.

To recover the active paths, we use the global routes variable obtained in the query phase. In case we find an active path, who uses the stopped switch, we use two kind of functions to try to find an alternative path.

The first function is called “limited_bi_directional_search”. In this case, the word “limited” refers to the extra limitations on the search (See Section 6.2.2 for more information).

If the limited search returns a positive result, the new path is calculated. If the result is negative, then we proceed to the second option called “Unlimited_bi_directional_search”. If between the source and destination nodes exist an alternative route, we must be able to reach it with a broader search from source to destination. If our modified BFS algorithm fails, we have another phase on the unlimited search, but before using this option, a change in the shortcuts must be done. See Section 6.2.3 for more information.

## 4.3.1 Recovery pseudo algorithm

**INPUT:**

```
    Shortcuts
        //list of strings, similar to links
        //obtained from pre-processing algorithm

    Active_switch
        //Dictionary of active switches
        //same as previous algorithms

    Links_information
        //dictionary of active links with information like:
        //source, destination, and active flag
    Global_Routes
        // list with strings type Src-V-Dst
```

**OUTPUT RESULT:**

```
    Updated_global_routes

    New_flow_route
        //function that renews the flow routes in the switches

    Updated_shortcuts
        //Only when necessary updates the shortcuts list
```

**PROCEDURE:** re_calculate_route[routes, sw_id]
//quick search for alternative fast routes

```
Previous_sw = routes[x - 1]

Next_sw = routes[x + 1]
      //value X is the index of switch_id in routes

Limited_Bi_directional_search[previous_sw, next_sw]

If Limited_bi_directional_search returns Null then

        Unlimited_Bi_directional_search[src, dst]

End

Return new_route
```

**PROCEDURE:** switch_leave_monitor[ev]
//built-in event for switch disconnecting

```
Switch_id = ev.switch.id //ev contains the information of the event

Active_switch.delete(switch_id)

For routes in Global_Routes do

    If switch_id in routes
    and switch_id is not routes.src
    and switch_id is not routes.dst then
                              //if the switch stopped is the source or destination,
                              //then re-routing is not possible

        New_route = Re_calculate_route(routes, switch_id)

        Global_routes ← New_route

        Install_flow_route(New_route, source, destination)

    End

End
```

# Chapter 5

# Algorithm Analysis

In this chapter we describe the results of the implementations to the algorithms from Chapter 4 and perform an analysis to understand how the algorithms work, the constraints, and propose a solution. As described previously in Section 4.2.1, the scope of these experiments is to propose and test our algorithms for a general environment to provide a theorical way to recover communications. To do this, we apply our algorithms in multiple types of networks and situations and here we provide the main scenarios.

## 5.1 Contraction Hierarchies Implementation

To fully understand the logic behind the "why the shortcuts and node order are created that way", we implement the CH algorithm in multiple topologies. One of the most useful topologies is shown in Figure 11.

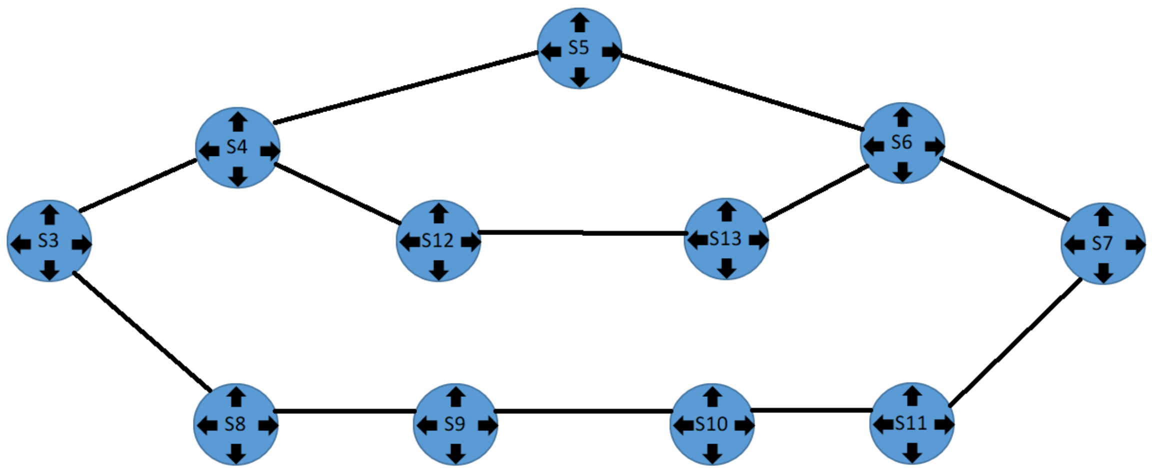


*Figure 11 Test Topology*

There are multiple reasons as to why we implement this topology. The main reason is that the switches offer multiple paths to reach a destination, this prove to be useful while testing our shortest path search. The second reason is to test the capacity of our algorithm to mitigate the problems with sending packages on topologies with loops. Finally, the topology is shorter than the Fat Tree topology which makes it easier to analyze. The analysis of a smaller graph allows us to identify the patterns of the algorithm. The patterns obtained gave us the idea to separate the graph into smaller segments to analyze the logic of each one. As an extra reason the shortcuts created while processing this topology are easier to draw in order to analyze the graph and realize the algorithms explanation.

After pre-processing the Figure 11 topology by applying the algorithm from Section 4.1.2, Figure 12 shows the result node order. The numbers inside the brackets represents the order in which the nodes were contracted. The contraction order means that during the query phase, a switch with a contraction order "*n*" will limit his search to other nodes with higher contraction number than its own. For example, switch 4 with node order 1 can search for switches number 12 and 5 with node orders 10 and 2 respectively, but not to switch 3 who has a contraction number 0. Figure 13 shows the results from the perspective of the shortcuts, these are also affected by the node order.

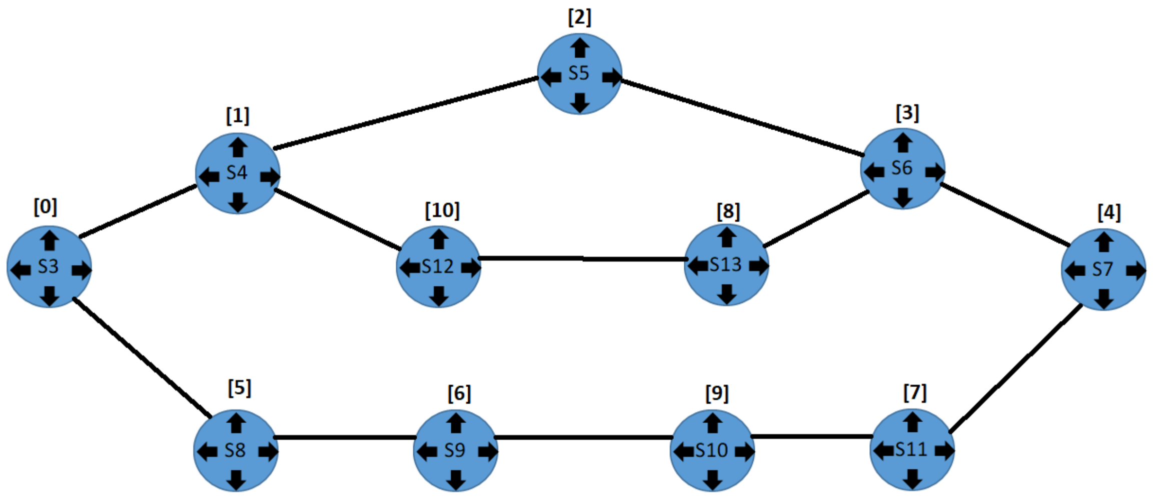


*Figure 12 Node Order*

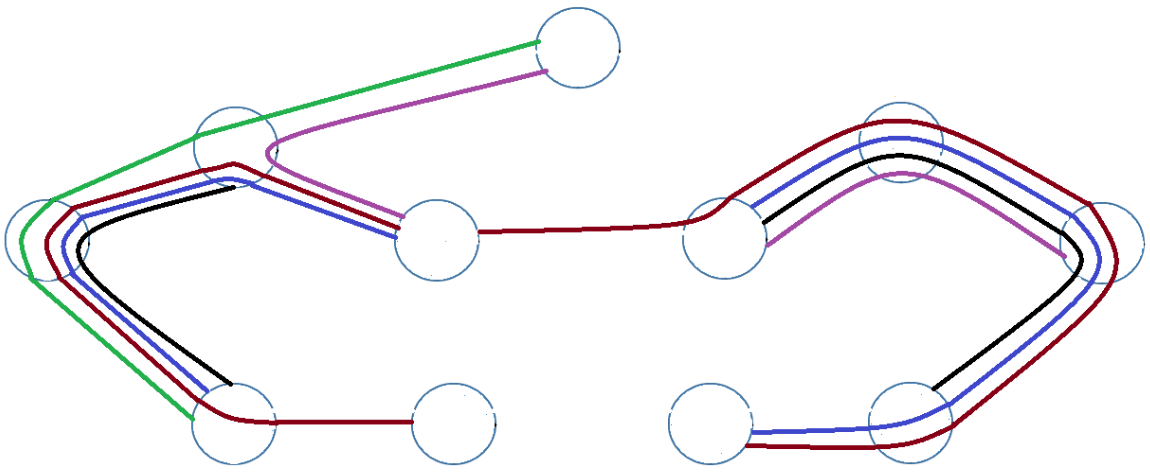


*Figure 13 Shortcuts*

## 5.2 Contraction Hierarchies Analysis

To analyze the performance of the CH algorithm, we needed a way to find and isolate the multiple paths and shortcuts between our switches and analyze the routing behaviors and constraints. As mentioned in the previous section, the best method we found to understand the CH algorithm, is to split our topologies into smaller sections. Each section analyzed represents a possible loop in the topology which contains one or more ways to connect our source and destination switches. Then we trace the direction of the node order and shortcuts on each sample. This method allows us to isolate at least two routes between the source and destination switches that belong to the same section and analyze if our CH configuration allows our query algorithm to obtain an alternative route if the original is interrupted. The results are as follows.

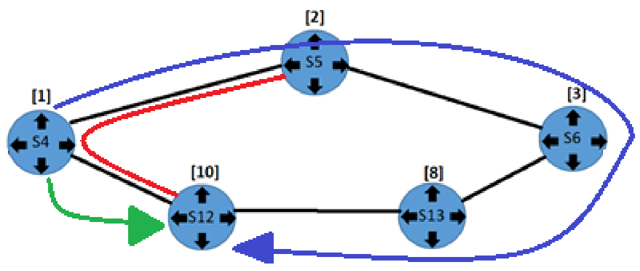


*Figure 14 Top Loop*

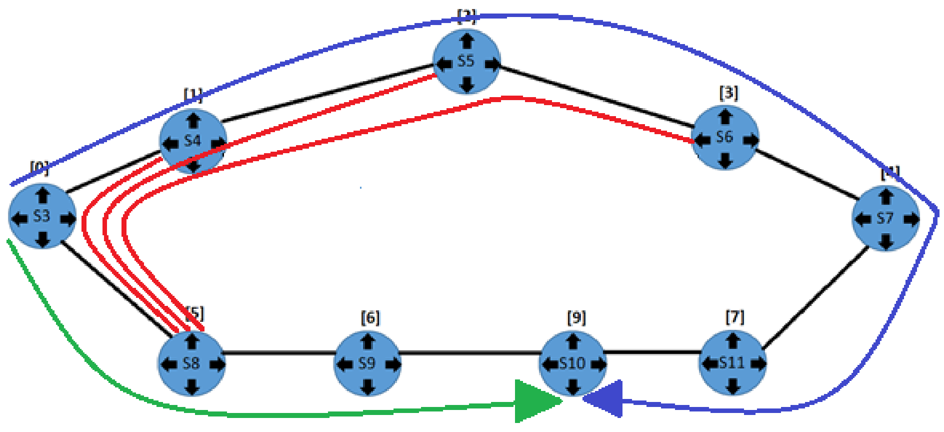


*Figure 15 External loop*

Figures 14 and 15 represent the top and external loops of the topology. Each loop has a local minimum, switches 4 and 3 respectively. In these cases, the loops formed contain only two search directions. If we follow the possible search paths from the local minimum and trace a line until the end, we will find the local maximum, switches 12 and 10 respectively. The external blue and green arrows represent the start, direction, and end of the searches that can be done with the links in those loops. The internal red lines represent the shortcuts previously obtained that involve only switches in that loop.

While making a search from any switch to any other switch, we always try to obtain the shortest possible route. The analysis of Figures 4 and 5 indicates that while searching for our path, the bi-directional search will follow one of the two paths until the search reaches a local maximum. In case the shortest path needs to go through the local minimum, the nodes can make use of the shortcuts to reach their destination.

The last loop is a different case because of its composition. As seen in Figure 16, the lower loop consists in two local minimums (switches 3 and 6) and two local maximums (switches 12 and 10) which produces shortcuts on multiple sides to move across the local minimum. Nevertheless, the normal algorithm still works for these cases and the number of shortcuts increases the speed of the path calculation.

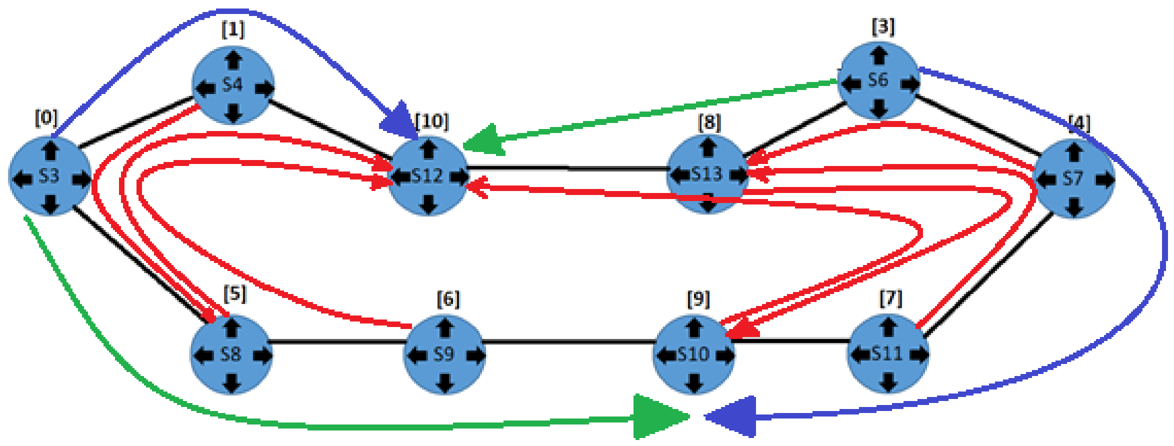


*Figure 16 Lower Loop*

After separating the previous topology and understanding the logic behind it, we proceed to do the same with the fat tree topology. Figure 17 represents a numbered version of the topology explained in Chapter 3. This topology is more complex, but the principles are the same. First, we identify a series of simple loops with simple patterns by applying the CH pre-processing and separating the loop sections. As the topology has multiple similar loop structures between its different layers, we will only show one example of each different structure. Nevertheless, the switches involved in the other structures will be described.

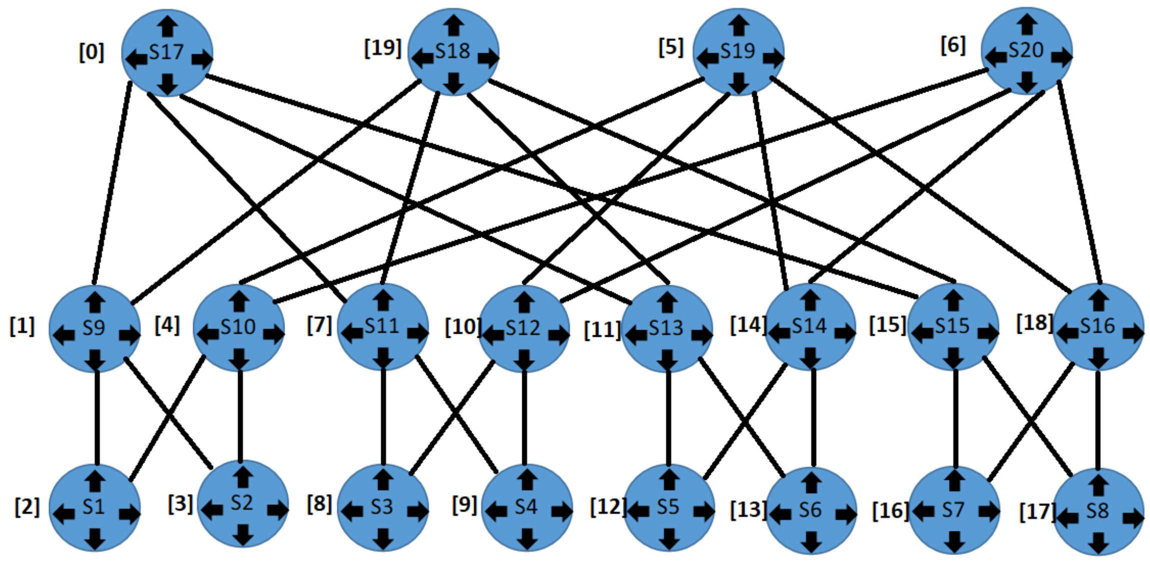


*Figure 17 Fat Tree with Node Order*

In Figure 18 we can see the shortest loops in the fat tree topology. Segments a, b and c are simple because there is only one minimum, one maximum and no shortcuts. Similar patterns to Figure 18a are repeated 4 times on the links between the aggregation and edge layers of the topology (the different layers are explained in Section 3.3). The pattern in 18b is repeated 6 times with the switches 17 and 18 and combinations of switches 9, 11, 13 and 15. Finally, the pattern in Figure 18c is repeated three more times with the switches 10, 19, 20 and either 12, 14 or 16.

The pattern in Figure 18d also contains only 4 switches, but the difference is that it contains two maximums and two minimums. As analyzed before, this requires a shortcut connecting both maximums. The pattern in Figure 18d is repeated 3 times with the switches 19 and 20, and two more switches between 12, 16 and 14.

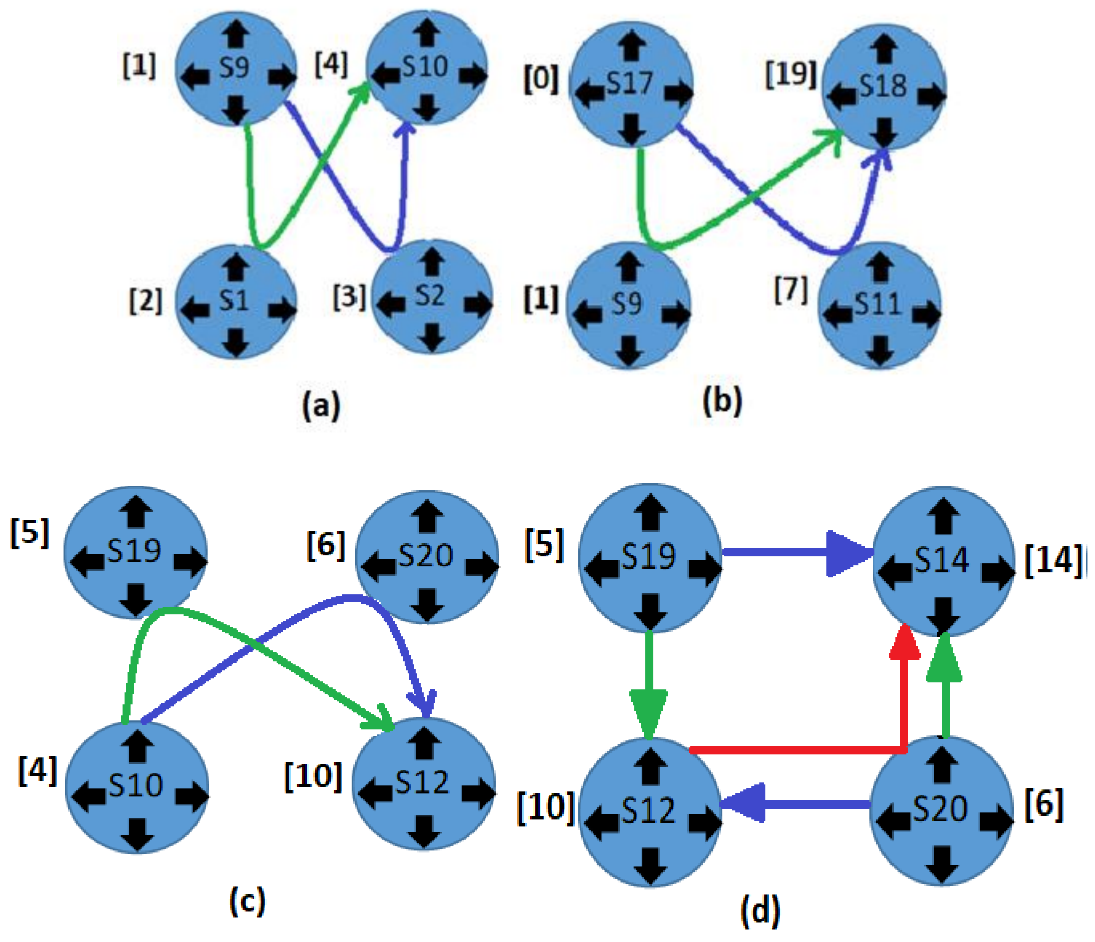
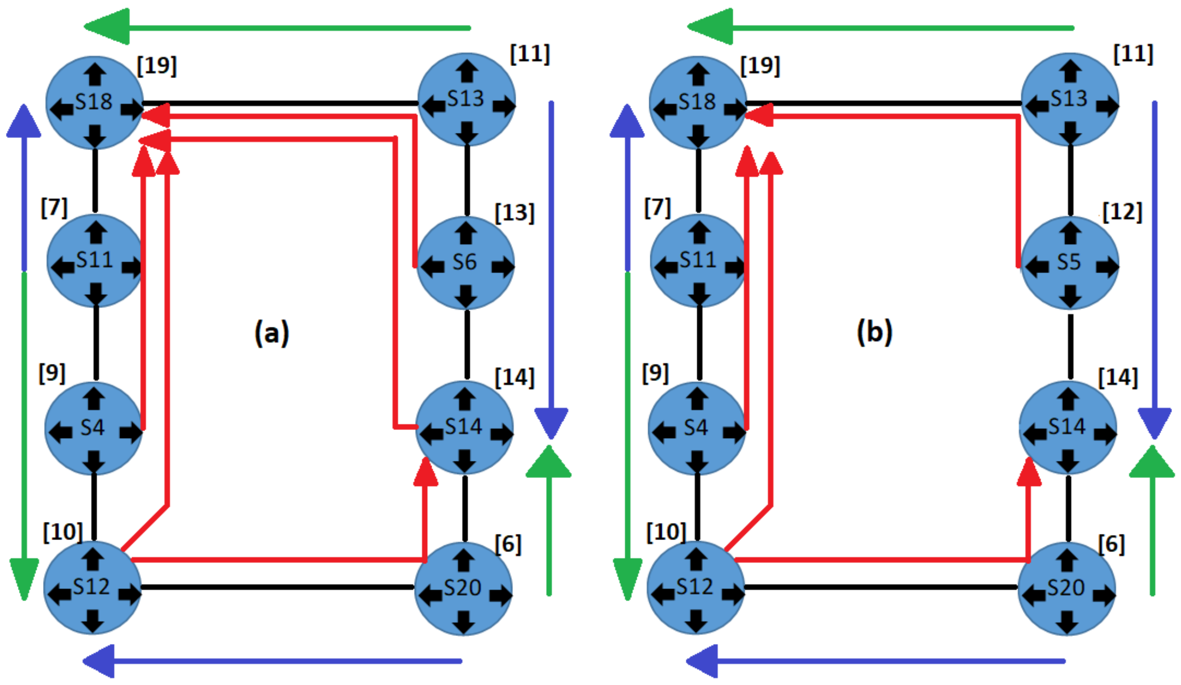
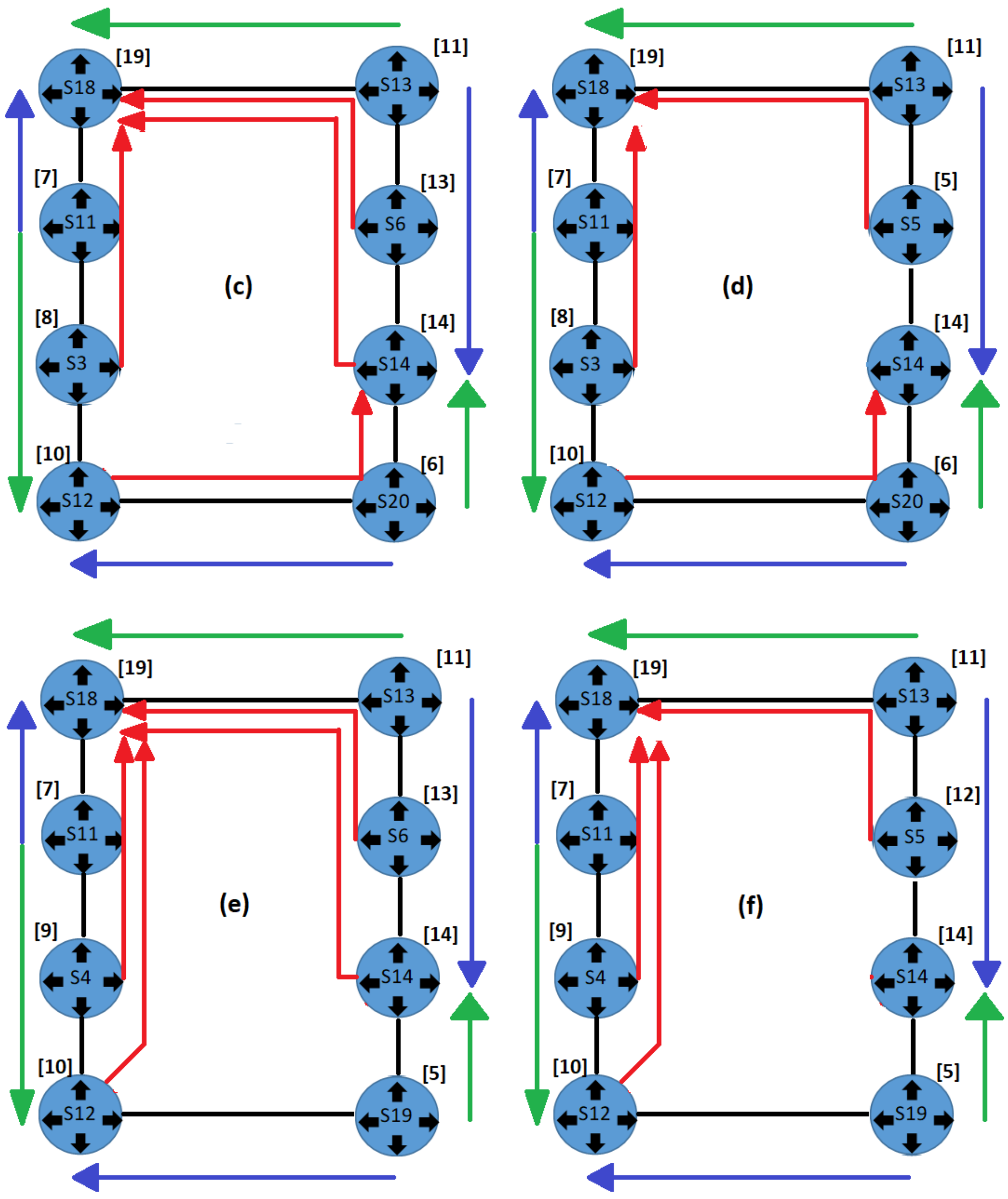


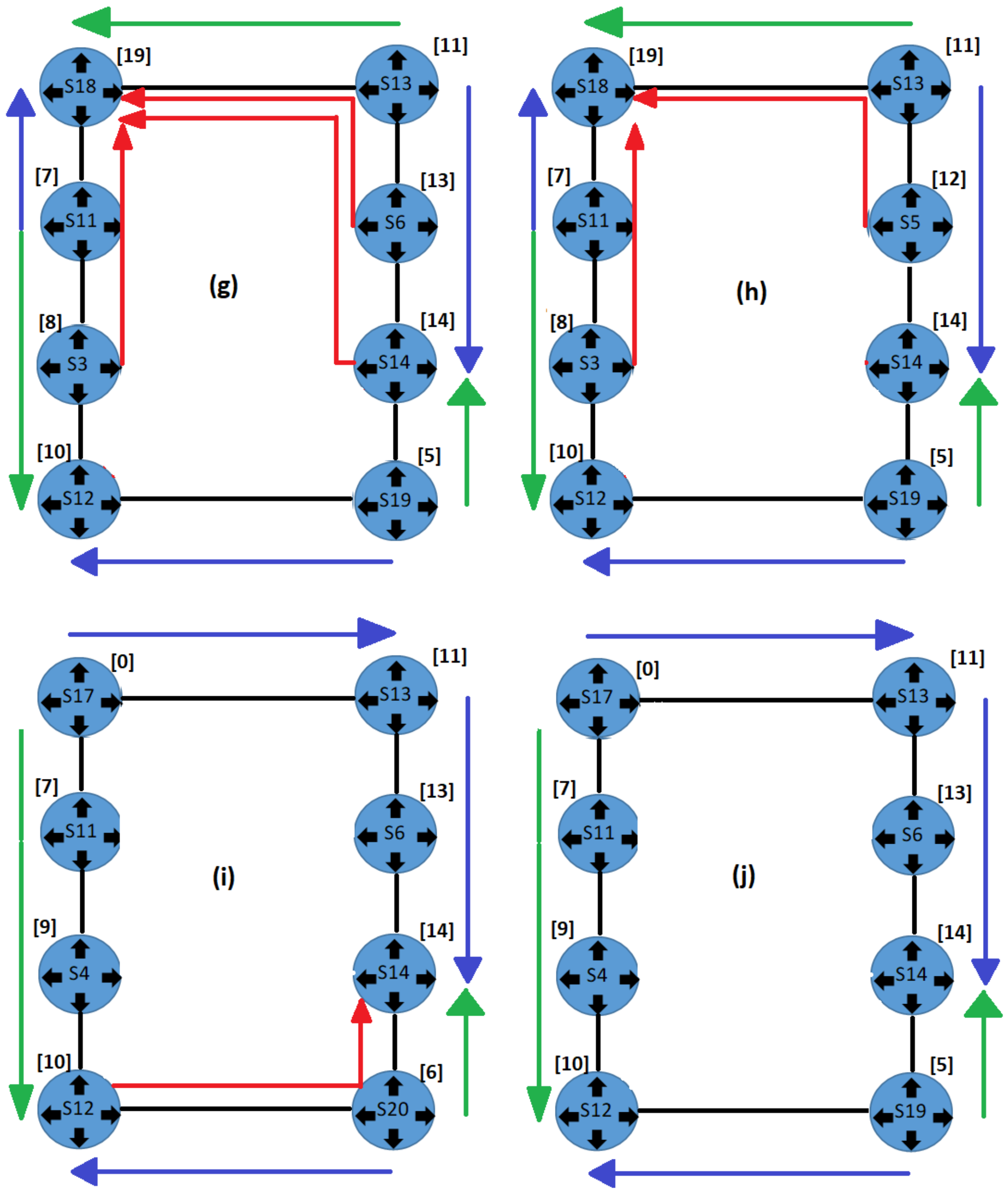
*Figure 18 Fat tree small loops*

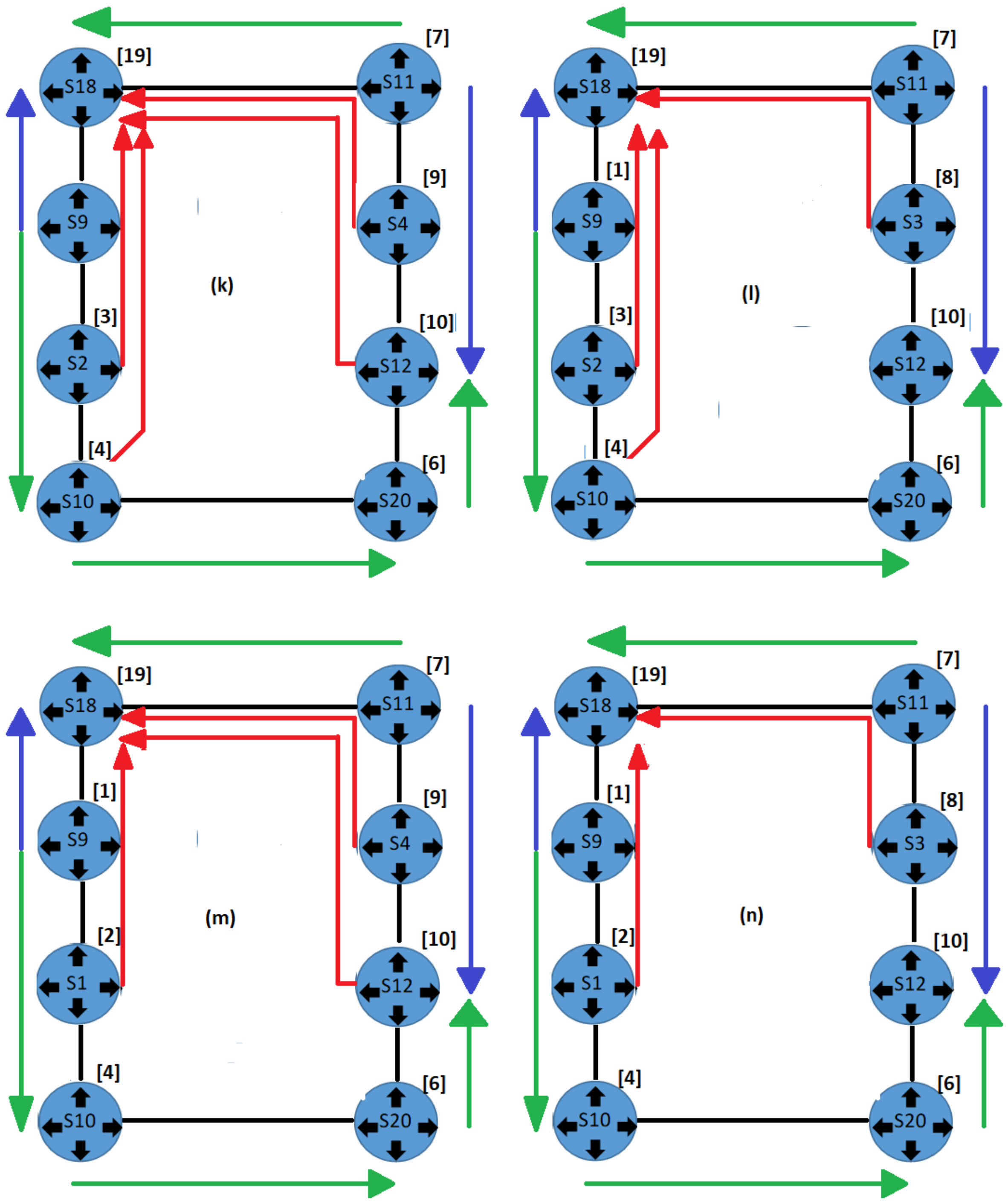
After isolating the small loops of Figure 18, the complexity increases for the remaining part of the structure analysis. Due to the links patterns of the lower layers of the fat tree topology, the following loops which connect most of the nodes involve all the layers of the topology (the three layers explained on Section 3.3.1). While isolating the following loops, the multiple possible shortcuts and path combinations caused multiple variations even on similar structures creating multiple types of similar loops. Next, we will show the multiple segments we found but we will omit those with repeated shortcuts and links flow.

As we can observe in Figure 17, the nodes 17 and 18 as well as nodes 19 and 20 have similar link patterns. Thus, the following loops are a combination of the links connecting node 17 with 19 and 20, or node 18 with 19 and 20. When node 17 tries to connect with either 19 or 20, it must go to a switch in the aggregation layer (9, 11, 13 or 15), then to the edge layer (1, 2, 3, 4, 5, 6, 7 or 8) and return to a different switch on the aggregation layer (10, 12, 14 or 16) and back to the core. For example, in Figure 19a, we connect switch 18 and 20 of the core layers. The connections of 18 and 20 with the aggregation layer are 18-11, 18-13, 20-12 and 20-14. Nodes 11 and 12 have at least one mutual node in the edge layer like 4, and 13 with 14 can be connected through 6 in the edge layer thus forming a loop. Finally, we add the possible shortcuts involving the nodes on each loop and the direction of the node order. Following the same logic, multiple loops are formed involving different combinations of the same shortcuts as shown next.




(c)
S18
S13
S11
S6
S3
S14
S12
S20
[19]
[11]
[7]
[13]
[8]
[14]
[10]
[6]
(d)
S18
S13
S11
S5
S3
S14
S12
S20
[19]
[11]
[7]
[5]
[8]
[14]
[10]
[6]
(e)
S18
S13
S11
S6
S4
S14
S12
S19
[19]
[11]
[7]
[13]
[9]
[14]
[10]
[5]
(f)
S18
S13
S11
S5
S4
S14
S12
S19
[19]
[11]
[7]
[12]
[9]
[14]
[10]
[5]


(g)
S18 [19]
S13 [11]
S11 [7]
S6 [13]
S3 [8]
S14 [14]
S12 [10]
S19 [5]
(h)
S18 [19]
S13 [11]
S11 [7]
S5 [12]
S3 [8]
S14 [14]
S12 [10]
S19 [5]
(i)
S17 [0]
S13 [11]
S11 [7]
S6 [13]
S4 [9]
S14 [14]
S12 [10]
S20 [6]
(j)
S17 [0]
S13 [11]
S11 [7]
S6 [13]
S4 [9]
S14 [14]
S12 [10]
S19 [5]



(k)
S18 [19]
S11 [7]
S9
S4 [9]
S2 [3]
S12 [10]
S10 [4]
S20 [6]
(l)
S18 [19]
S11 [7]
S9 [1]
S3 [8]
S2 [3]
S12 [10]
S10 [4]
S20 [6]
(m)
S18 [19]
S11 [7]
S9 [1]
S4 [9]
S1 [2]
S12 [10]
S10 [4]
S20 [6]
(n)
S18 [19]
S11 [7]
S9 [1]
S3 [8]
S1 [2]
S12 [10]
S10 [4]
S20 [6]

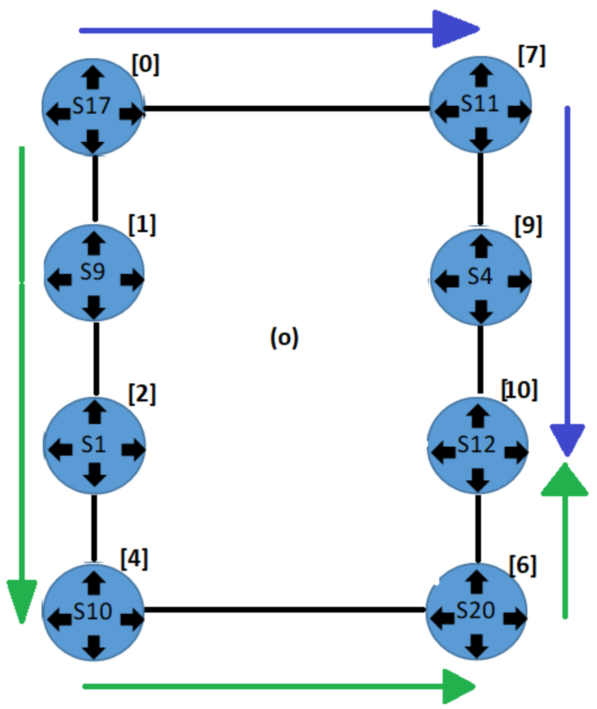


*Figure 19 Loops obtained from the analysis of the fat tree topology*

After separating the different loops in the fat tree topology, we can observe similar cases to those obtained from Figures 4, 5 and 6. Something worth mentioning about these new loops is that most of them have three minimum and three maximum nodes but they still follow the pattern for the searches. As we can observe in Figure 19a, three of the 5 shortcuts we have can be used to connect the 3 maximums when needed to reach our destination. Finally, the number of shortcuts decrease as we change the nodes 18 and 20 for the 17 and 19 respectively, and nodes 4 and 6 for nodes 3 and 5. This decrease of shortcuts corresponds to similar structures with nodes in lower node orders. We can conclude that the algorithm gives preference to searches with higher node orders that accelerate the procedure with the shortcuts.

## 5.3 Recovery Phase Analysis

Up to this point we have a functional implementation of the CH algorithm and the modified version of the bi-directional search. Nevertheless, when performing a search and stopping at least one of the switches in the route, we have multiple scenarios.

In Figure 14 we can observe a shortcut from 5 to 12 (5-4-12). While tracing the route from 5 to 12, this shortcut is useful because it allows us to do a fast search in a single step. The main problem comes when switch 4 is stopped. In this case, we can see that there is another route through 5-6-13-12. The new route has two problems involved with the CH algorithm. The first problem is that the previous shortcut calculation does not contribute towards the calculation of the fastest route. The second is that, contrary to its purpose, the node order prevents node 12 to search towards node 13 because it has a lower order value. This means that node 5 must loop through all the remaining elements by itself, which converts the algorithm into a normal BFS.

In Figure 15 we have a similar case than the previous one. In this example, if we try to go from switch 6 to 8 and a switch in the shortcut 6-5-4-3-8 fails, we will have no gain with respect to the shortcuts. Nevertheless, the bi-directional search does still contribute towards speeding up the path calculation. Node 8 can still search through nodes 9 and 10, and node 6 can search through nodes 7, 11 and 10 which would reduce the number of steps from 5 (in the original BFS) to only 3. In the worst-case scenario, nodes like 3 and 4 contain most shortcuts in this loop, in cases when these types of nodes are stopped, the event will nullify most of the shortcuts available.

Fortunately, in cases like the one shown in Figure 16, the biggest advantages of the chosen algorithms arise. As mentioned before the loop in Figure 16 contains two local minimum nodes, which produces shortcuts on two different areas of the loop. In the case of communication between nodes 12 and 9 for example, and the interruption of the path between them, if the path 12-4-3-8-9 gets interrupted, we can still calculate another path between our source and destination while optimizing the search with the advantages of CH. In general, the pattern on Figure 16 is repeated, in Figure 19a when we have three maximums and three minimums the shortcuts improve the communication to the different maximums and through the minimums.

# Chapter 6

# Experiments and Results

This chapter describes the constraints we found during our experiments and our proposals to solve them. In [Gei12], Geisberger et al. contemplate a dynamic scenario due to the constant changes on road networks. In this work, we believe that contemplating a dynamic network is the right approach due to events that can occur on computer networks. Nevertheless, our approach is different in both, the type of events that we contemplate, and the way we tackle the problems.

## 6.1 Query Constraints

As explained before, considering our network limitations and lack of real case scenarios, our proposed network does not contemplate dynamic weights. This means that changes on links weights due to changes on the networks are harder to contemplate, so we decide to limit our events to disconnections. Next, as described on Section 2.1.1, Shamim et al in [Sha18] did some work on link aggregation and used SDNs to implement an option during unforeseen events on the links. Due to the individual nature of network devices, during our research we often found a network element-based concern about the switches proper communication during link disconnection or switch disconnection. In computer networks, both cases (a link or a device disconnection) need to be effectively informed to other switches and in SDN cases to the controller. Thus, our approach was based on computer network circumstances and the algorithm’s performance analysis performed in Chapter 5 which led us to our approach.

From Chapter 5 we can observe that the searches from source and from destination of the bi-directional search tend to converge into the local maximums. The main problem comes when the path to the maximums or the path between the maximum is interrupted. To exemplify these cases, we found mainly 3 examples.

### 6.1.1 Direct Query Between Local Maximums

The first problem example is in the example from Figure 16. We found a problem while tracing the paths between the local maximums (10 and 12). Usually, local maximums in a loop cannot move to other nodes through normal links because the adjacent nodes have a lower node order. To solve these cases, CH generates a shortcut between the local maximum to connect the nodes where the searches converge. The problem comes when this shortcut is cut by one of our unforeseen events, then we would need to find another path between nodes 10 and 12. In these cases our search won’t be able to generate another path.

### 6.1.2 Query Stops on Different Local Maximums

A similar problem occurs when in a recovery search one of the nodes tries to contact a local maximum i.e. in Figure 16, node 7 contacting node 10 through 7-11-10. In this example, when 11 fails, 7 only contains one path that leads to the local maximum 10, a shortcut to 13, and 13 can reach 12. 10 is uncommunicated because its shortcuts were cut by 11. While 7 can still perform some movements, it can only reach 12, thus producing a variation of the problem in Section 6.1.1 and a disconnection between 7 and 10.

### 6.1.3 Unnecessary Elements in the Calculated Routes

Finally, while splitting the graph into the multiple loops helps us to analyze the individual pieces, we must not miss the whole picture. In Figure 14 while connecting 6 and 12 through 6-13-12, if 13 stops, it might seem like the connection between 6 and 12 is lost such as in our previous example. While looking at Figure 6, node 6 is also a minimum and has no search capabilities. Nevertheless, while looking at Figure 15, node 6 has a shortcut to 8 (through 6-5-4-3-8), note that node 12 is not in this figure. Finally, Figure 16 contains a route from 8 to 12 that does not involve node 13, thus, the route from 6 to 12 is possible (6-5-4-3-8-3-4-12). The result from the query is long and contains unnecessary nodes for the final route while moving twice through nodes 3 and 4 and having 8 as an unnecessary node. While the final route should be 6-5-4-12, the recovery node was unable to identify these loops. So, to improve the final route, a final route-processing method was implemented.

## 6.2 Proposed Solutions

### 6.2.1 Unnecessary Element Search

While calculating the routes we found problems with unnecessary loops like the one presented on the final example of the previous Section 6.1.3. This issue was easily solved by adding a little procedure on the route calculated.

**PROCEDURE:** Eliminate_final_route_loops[final_route]

```
For x in range(len(final_route)) do
    For y in range(x + 1, len(final_route)) do
        If final_route[x] equals final_route[y] then
            New_route = Delete_elements_between_x_and_y_plus_y()
        End
    End
End
```

This new procedure can be implemented while calculating any of the final routes through the implementation to avoid unnecessary loops.

“Eliminate_final_route_loops” iterates through the elements of the calculated route and eliminates the intermediate elements when it founds the first match. In the previous example, we can observe that the recovery route 6-5-4-3-8-3-4-12 contains two unnecessary loops. In this case, X iterates through the elements starting from 6, Y iterates through the rest and compares them. The “if” comparison will not return positive until x reaches the value 4 (in the 3rd position) and Y reaches 4 (in the 7th position), then the procedure will delete the elements in-between (3-8-3) this will also eliminate the internal loop. Finally, the only step towards our final route is to delete either of the repeated values (node 4) and our route will be ready to be installed.

### 6.2.2 Alternative Switch Search

Previously in Section 4.3.1, we mentioned two types of recovery functions. After analyzing the logic behind the pre-processing (Chapter 5), now we can analyze the reasons for these functions. The first one named "Limited_bi_directional_search" corresponds to examples when it would be faster to find an alternative route to the stopped switch, rather than a completely different one.

First, we calculate the previous and next indexes of the switch stopped in a route, then we realize a search without going back to the source and destination switches of the original route. This makes sense when there are other routes connecting both switches and we can simply replace the lost switch in the fastest way possible. This also corresponds to how we implemented the algorithm because if there are two routes between the same points, it only contemplates the shortest. In case the two possible routes have the same length, then it ignores the first one and only adds a shortcut through the second one. The equations to calculate the values for the limited search were presented in the procedure "re_calculate_route" in Section 4.3.1, the rest is a simple bi-directional search using the input values for the limited search.

For the particular case of the "Limited_bi_directional_search" we will ignore the node order during the search, this will return results like node 17 (node order 0) instead of node 18 (node order 19) in Figure 17, as long as the resulting node can replace the event node. Finally, we include a max_len value to limit the number of iterations on the search. While using 1 as max_len can be useful to replace nodes like 18 and 17, in Figure 14 we can increase max_len to replace 5 in 4-5-6 with a short route through 4-12-13-6. This value can be changed according to our needs, but higher values might not return better routes.

**PROCEDURE:** Limited_bi_directional_search[previous_sw, next_sw]

```
Level = 1
Visited = {}
Visited = {start: 0, stop: 0}
For i = 0, 1, 2, … max_len do
    For node in visited do
        If node.level equals level - 1 then
            Adjacent = adjacent_nodes[node]
            For adjacent in Adjacent do
                if adjacent in visited and source_of
                visited.node_route not equal to source_of
                adjacent.node_route
                    alternative_route ← Calculate_alternative_route()
                    Return Calculate_alternative_route()
                Else
                    Visited ← {route_to_adjacent: level}
                End
            End
        End
        Level = level + 1
    End
End
Return false
```

### 6.2.3 Recovery Search

When replacing the switch of the event is not possible, we must re-calculate the complete route. Here is when the second option mentioned in Section 4.3.1 “unlimited_bi_directional_search” comes in. In this case, the word unlimited refers to the lack of restrictions of the algorithm while we search for a different path to take, the algorithm is separated into two phases.

During the first phase, we perform a normal bi-directional search with the available nodes (node order restrictions and shortcuts included), to try to locate an alternative route (if the route exists within the CH restrictions). If the search reaches the maximum values available but these two maximums cannot communicate, then we implement the second part of the algorithm.

The second phase of the unlimited search activates when the shortcut between the maximums is cut as seen in Sections 6.1.1 and 6.1.2, but there is still an alternative path for our route. Here we must find the alternative route between our maximums without the restriction of the CH algorithm, this will restart the communication while maintaining the CH node order and shortcuts.

While tracing the new path between the local maximums, we perform what we call an unlimited search. To find the new route, we can calculate a new path by using links and available shortcuts. In this case, we can go to any node despite the node order. This will guarantee a new shortcut through an available route and a new connection between local maximums. This new connection can be added to the shortcut variables to help other searches that include these nodes. This method is more effective in comparison to a normal bi-directional search from source to destination because by using the normal algorithm we obtain a faster path, but we do not contribute to calculate others. By using our method, the shortcut can improve other path calculations.

**PROCEDURE:** unlimited_bi_directional_search[src, dst]

```
Level = 1
Visited = {}
Visited = {src: 0, dst: 0}
For node in visited do
    If node.level equals level - 1 then
        Adjacent = adjacent_nodes[node]
        For adjacent in Adjacent do
            If adjacent.order_number < node.order_number then
                Ignore lower order nodes
            Else if adjacent in visited and source_of
            visited.node_route not equal to source_of
            adjacent.node_route
                Global_routes ← Calculate_final_route()
                Return Calculate_final_route()
            Else
                Visited ← {route_to_adjacent: level}
            End
        End
    End
    Level = level + 1
End
```

```
Src_maximum, dst_maximum = Calculate_search_maximums()

Level = 1

Visited = {}

Visited = {src_maximum: 0, dst_maximum: 0}

For node in Visited do

    If node.level equals level - 1 then

        Adjacent = adjacent_nodes[node]

        For adjacent in Adjacent do

            if adjacent in visited and source_of
            visited.node_route not equal to source_of
            adjacent.node_route

                Shortcuts←shortcut_from_maximum_to_maximum()

                unlimited_search_phase_one()

            Else

                Visited ← {route_to_adjacent: level}

            End

        End

    End

    Level = level + 1

End
```

This last option can create loops as the ones mentioned in Section 6.2.1, in these cases we use the proposed "Eliminate_final_route_loops" procedure on the same section to obtain the final route.

## 6.3 Results

To test our algorithms, we use the implementation from Chapter 4 and the topologies from Section 3.3. The pre-processing algorithm is automatically activated once all the switches and links connect to the controller.

Our general setup consists on several steps. First, after the pre-process has been completed, we select two specific switches which connecting route being stopped can generate our specific constraints from Section 6.1. Then, we send 20 packages between the source and destination. To make the times more constant, we waited for the first 10 packages to be sent. After the first 10 packages, we stopped specific switches of the routes that produced variations of the problems presented on Section 6.1. Finally, we let our program find and run the corresponding solutions from Section 6.2 to fix and re-install the routes. Each graph in this section represents a step of the proposed solution and the time the different algorithms take to fix the routes. The algorithm follows a specific order. First it tries to fix the original route by using the algorithm on Section 6.2.2 ("limited_bi_directional_search"). If this procedure fails, then it tries to obtain a different route by using the first phase of the algorithm in Section 6.2.3 ("unlimited_bi_directional_search"). In case it fails, then the second phase of the algorithm in 6.2.3 tries to fix the infrastructure and calculate a route.

To optimize the survivability of the transmission, we focus on providing solutions in a specific order to optimize the recovery time. The first set of experiments and Figure 20 represent the cases when the route can be fixed by using the "limited_bi_directional_search" from Section 6.2.2. In this first series of experiments, we stopped switches that affect active routes and that can be easily replaced by alternative similar routes without re-calculating the complete route. For example, in the topology of Figure 12, while connecting switches 3 and 7 (route 3-4-5-6-7), we stopped switch 5. The algorithm was able to calculate a replacement route (like 3-4-12-13-6-7) if the replacement route exists within two simple limitations. First, the search for the alternative route must not go back to the original source and destination (3 and 7). Second, the iterations for the search are lower than our max len value of 3 (see Section 6.2.2 for more examples). First, we implemented the general setup and stopped the switches as mentioned in the previous example. Our graph (in Figure 20) shows the average time vs package number behavior amongst 10 runs. We can observe that the selected switches were stopped after the 10th package and as shown in the respective Table 1 numeric values, we experience an average delay of 33.06 ms for the 11th package.

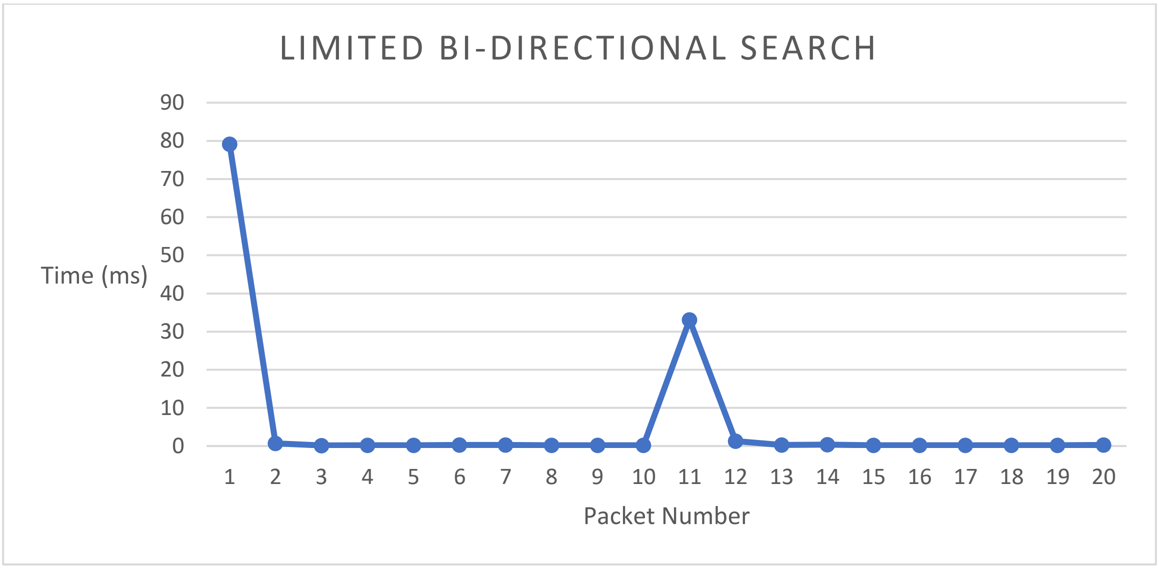


*Figure 20 Alternative Switch Recovery*

| Packet | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time (ms) | 79.17 | 0.683 | 0.1599 | 0.2046 | 0.2249 | 0.274 | 0.293 | 0.213 | 0.2008 | 0.2003 |

| Packet | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time (ms) | 33.06 | 1.2557 | 0.2733 | 0.4003 | 0.2327 | 0.2432 | 0.21 | 0.2341 | 0.24 | 0.2763 |

*Table 1 Limited bi-directional search values*

When the first algorithm fails to fix the route, the program access the second algorithm ("unlimited_bi_directional_search" Section 6.2.3) to try to calculate one of the alternative routes.

The next part of our recovery process corresponds to the first phase explained in Section 6.2.3. This process corresponds to the cases when a completely new route needs to be calculated but we can still calculate this route by using the benefits of the CH algorithm. We implement the general setup explained at the beginning of this section and stop the corresponding switches so our program runs the first phase of the recovery search. A specific example of a switch for this section is seen on Figure 17 when switch 1 connects with switch 3 (route 1-9-18-11-3). If switch 9 is stopped, switches 1 and 18 cannot be connected by using the previous procedure. In this case, we need to recalculate the complete route. Nevertheless, we can still calculate the route by using the benefits of the CH pre-processing. As seen in Figure 19n the green arrows from switch 1 lead to switch 12 which has connection with switch 3 thus generating the route 1-10-20-12-3. This experiment has multiple variations as we can reproduce the same process by stopping either switch 9 or switch 11. In the graph of phase two (Figure 21) similar to phase one, we observe the average time-package behavior of nine test runs following the general setup with switches being stopped after the 10th package. In this case, in the corresponding Table 2 numeric values, we observe an increment of the average time up to 37ms. Considering the fluctuation of the times needed to restore the communication, a small increment on the average time was expected due to the extra process of route search.

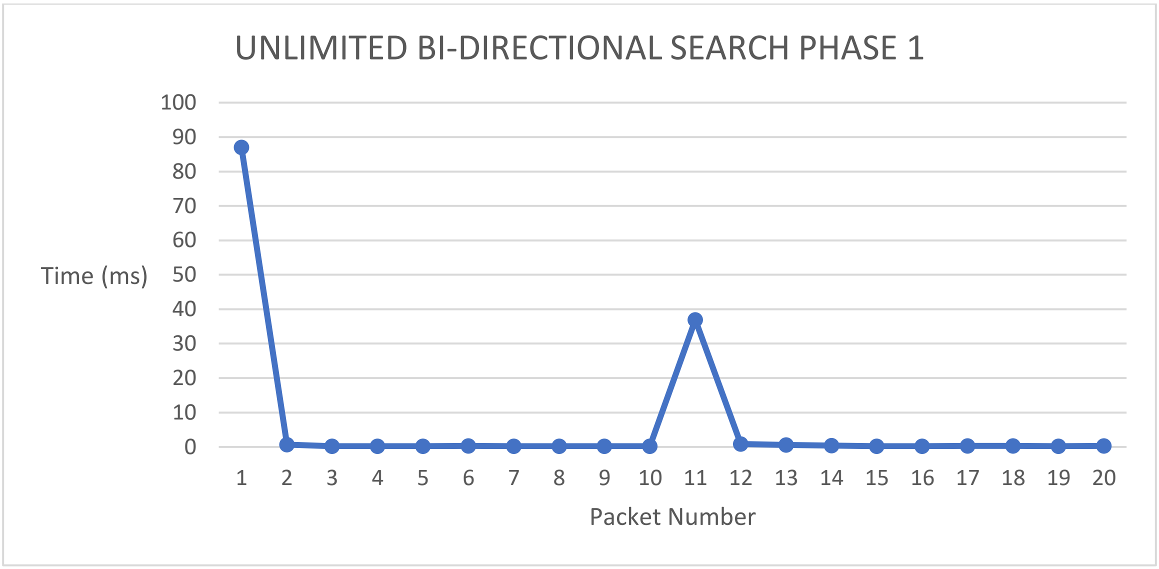


*Figure 21 Recovery search process one*

| Packet | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time (ms) | 87.0333 | 0.6771 | 0.2313 | 0.2316 | 0.2542 | 0.263 | 0.2147 | 0.2337 | 0.2138 | 0.217 |

| Packet | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time (ms) | 36.8888 | 0.8472 | 0.5348 | 0.4148 | 0.2175 | 0.2275 | 0.2825 | 0.309 | 0.2111 | 0.2594 |

*Table 2 Unlimited bi-directional search phase 1 values*

When both the processes described previously fail, the second process (first part of the "unlimited_bi_directional_search") returns the local maximums reached (i.e. 12 and 10 from the examples in Sections 6.1.1 and 6.1.2), then we proceed to the final phase of the recovery process, this phase corresponds to the final function described in Section 6.2.3. In this phase, the path cannot be re-calculated using the modified BFS algorithm (available shortcuts and node order) but we might still have a path available. For a specific case of this phase, we can consider the case scenarios described in Sections 6.1.1 and 6.1.2 in which our searches reach different local maximums and we must reconnect both before fixing our route. We use the output of the previous phase (12 and 10) as the input values for the final procedure.

The final phase is executed automatically only after the previous phases failed, because of the time needed to try the previous phases is still present, we expected a time increase in the package transmission.

Figure 22 shows the average of eight experiments. Similar to the previous experiments we use the general setup and stopped the switches as explained in Sections 6.1.1 and 6.1.2, the transmissions were interrupted after the 10th package. In the Table 3 numeric value representation, we can observe an average time increase up to 50.58ms, which is expected as the final phase involves two different searches.

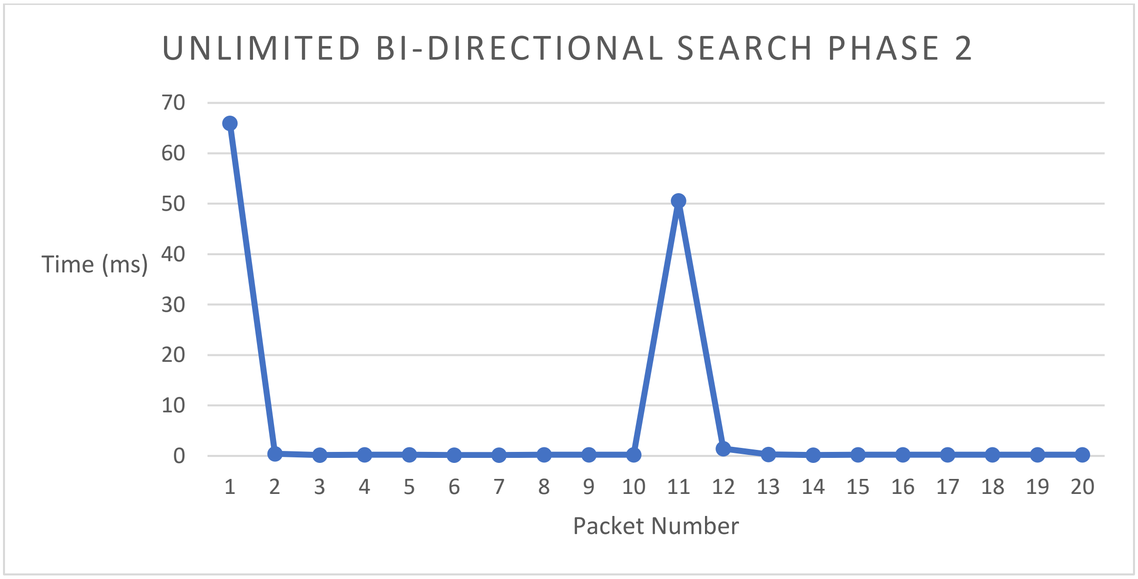


*Figure 22 Recovery search final phase*

| Packet | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time (ms) | 65.975 | 0.4391 | 0.1912 | 0.2108 | 0.2425 | 0.1663 | 0.1805 | 0.2083 | 0.2273 | 0.244 |

| Packet | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time (ms) | 50.5875 | 1.4142 | 0.2923 | 0.1928 | 0.2668 | 0.2273 | 0.248 | 0.2652 | 0.2183 | 0.2307 |

*Table 3 Unlimited bi-directional search phase 2 values*

# Chapter 7

## Conclusions and Future Works

After researching and implementing our ideas, we can say that there are two essential points on our research.

First, the analysis of other research which lead to the idea of comparing computer and road networks. By reading other works, we often found that most implementations of interdisciplinary algorithms were not the full algorithm but “modified” versions of them. After reading a few articles we observe that those modifications were mostly to adapt to the necessities and the types of graphs where they were needed. While the comparison of Computer and road networks show many useful similarities that allow us to implement the Contraction Hierarchies algorithm, we believe that the key point was to identify the differences of the networks in order to fully adapt other algorithms to our requirements. To find these differences, we believe that dedicating some effort on the comparison was essential.

Second, the implementation of the Software-Defined Networks paradigm facilitates the coordination between network devices and is essential to implement centralized algorithms to coordinate the network. Our setup allows us to analyze the advantages of the CH algorithm on the network and plan the re-routing procedures. The selected pre-processing procedures created a logic on the nodes order and shortcuts that allow the algorithm to create routes as long as there is a path available, and in most cases to improve the path selection with the shortcuts and node order even when some elements were disconnected.

In general, we observe an expected increment on the times involved on every phase of recovery. The times obtained at the beginning of the simulation oscillate between 50ms and 120ms. Although the times above 100ms were few, we still counted these experiments as we believe that these results did not affect the re-routing process and we consider that multiple factors are involved while obtaining these times. Some examples of these factors are the following:

- First, as this is the first time doing these kinds of experiments, we wanted to make sure the data obtained before the recovery process was as accurate as possible. To do this, our code includes many validations and console outputs to rectify our data.

- Next, at least 10ms were lost during the code involved on the thread management. This time loss was entirely on purpose and more information is provided later on this chapter.

- Finally, we need to consider the lack of resources and our hardware factor. We believe that our environment for testing being a single virtual machine on a laptop is a factor to consider while running multiple threads.

Nevertheless, and taking into consideration the limitations of our implementation, we believe that out results during the re-routing are satisfactory due to the following reasons.

- First, similar to the routing process, the 10ms time loss while managing the threads is also involved on the re-routing process. This means that the final times of our procedure should be at least 10ms faster when implemented on a different setup.

- Our recovery process fits our implementation of the Contraction Hierarchies algorithm and finds the routes as long as they exist.
- While we can appreciate an increase of the transmission times during the recovery process, these times were never higher than those of the original routing process, the connection was not interrupted, and during our experiments the packages were not lost.

As our approach is based on the survivability of the network, we believe that one of the advantages of our method is to avoid the re-contraction of the nodes as much as possible to optimize the route re-calculation. In a large network, a server needs to calculate multiple routes as fast as possible and fix the current ones in the shortest amount of time to keep the communication active and reduce delays. To observe how our algorithm behaves during transmissions, we focused on optimizing the re-routing procedure for cases when communication is active. Thus, we believe our multiple step approach is a good option to optimize time when the solution can be easier than re-calculating the whole route, and in case of complete re-calculation, it allows other searches to take advantage of the results obtained.

While our research was able to restore the communication between nodes, there is more work that can be done. As mentioned on Chapter 2, while centralizing the control layer in a single device improves the coordination between the network elements, there are some security aspects that need to be taken into consideration.

First, we created a single point of failure. After reading other research papers we conclude that there are other works focused on solving this problem. Thus, we believe that focusing on our problem is a good start for researching SDN communications.

In the future, we would like to do a performance test on our chosen controller with more data available. While Mininet and RYU controller allow us to setup some characteristics to simulate a complete network, we believe that more testing is required. Using the number of jumps as weight measure for the routes is sufficient as a general setup, but to add more realism to our implementation, more parameters are required. We can add more case scenarios depending on specific needs of an organization’s network like type of data, type of applications, extension of the network and topology, amount of data required to be transferred, etc.

On the Contraction Hierarchies algorithm side, as explained by its creators, we can implement multiple variations of the pre-processing method which affect the effectivity of the results. While we did not perform tests and analysis on all the different implementations, we would like to find how these changes influence the results of the pre-process and how would that differ from our analysis performed in Chapter 5.

While we focus on different areas (than [Gei12]) like:

- A different type of network.
- Put more emphasis on the nodes rather than the links.
- Focus on disconnections rather than other possible changes in the network, and
- Focus on fixing the whole routes rather than the shortcuts.

We also have some similarities, we tried to preserve the original node order as much as possible instead of re-contracting the whole graph and correct the query. Nevertheless, we believe that the differences in methods came from our desire to optimize the survivability of the network.

During the process of installing the flows on the network elements, we detected an inconvenience with Mininet. Our implementation can install multiple flows on a single packet_in[10] event. Nevertheless, we discover that during the flow distribution, the serialization of Mininet processes had problems while installing the flows on the network elements before the thread in execution sends the package to the next element, thus creating an extra unnecessary packet_in event. We controlled this problem using python thread functions but adding an extra delay time was necessary. To solve this, we believe that testing the performance on a different type of environment is essential.

In general, we were able to understand the performance of SDNs and to test our limitations. Our interdisciplinary setup allows us to implement our ideas and to analyze the performance of our algorithm to solve our specific problem. During our tests, the theorical analysis and positive results added to the fact that communication between hosts was not interrupted and only suffer small delays, encourage us to improve our setup and hope that this work can encourage the readers to expand our ideas.

[10] Packet_in is the name of the controller function that process the request for routes on the controller.

# References

[Ahn19] Ahn, S., & Choi, J. (2019). Internet of vehicles and cost-effective traffic signal control. Sensors (basel, Switzerland), 19(6). doi:10.3390/s19061275

[Alc14] D'Alconzo, A., Casas, P., Fiadino, P., Bar, A., Finamore, A., & 2014 International Wireless Communications and Mobile Computing Conference (IWCMC) Nicosia, Cyprus 2014 Aug. 4 - 2014 Aug. 8. (2014). 2014 international wireless communications and mobile computing conference (iwcmc). In who to blame when YouTube is not working? Detecting anomalies in CDN-provisioned services (pp. 435-440). IEEE. doi:10.1109/IWCMC.2014.6906396

[Aou17] Aouadj, M., Lavinal, E., Desprats, T., & Sibilla, M. (2017). Airnet: An edge-fabric abstraction model to manage software-defined networks. International Journal of Network Management, 27(6). doi:10.1002/nem.1983

[Bot12] Botta, A., Dainotti, A., & Pescapé, A. (2012). A tool for the generation of realistic network workload for emerging networking scenarios. Computer Networks, 56(15), 3531-3547. doi:10.1016/j.comnet.2012.02.019

[Bot13] Botta A., Donato W., Dainotti A., Avallone S., & Pescapé A. (2013). D-ITG 2.8.1 Manual. COMICS (COMputer for Interaction and CommunicationS) Group Department of

Electrical Engineering and Information Technologies University of Napoli Federico II. Retrieved from: ditg.comics.unina.it/manual/D-ITG-2.8.1-manual.pdf

[Cis19] Cisco. (2019). Cisco Visual Networking index: global mobile data traffic forecast update, 2017-2022 white paper. Retrieved from: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html#_Toc953327

[Cis19b] Cisco. (2019). Cisco visual networking index: Forecast and trends, 2017-2022 White paper. Retrieved from: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html#_Toc532256798

[Dew18] Dewanto, R., Munadi, R., & Negara, R. (2018). Improved load balancing on software defined network-based equal cost multipath routing in data center network. Jurnal Infotel, 10(3), 157-157. doi:10.20895/infotel.v10i3.379

[Dil02] Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., & Weihl, B. (2002). Globally distributed content delivery. Ieee Internet Computing, 6(5). doi:10.1109/MIC.2002.1036038

[Flo17] Flores Moyano, R., Fernández, D., Bellido, L., & González, C. (2017). A software-defined networking approach to improve service provision in residential networks. International Journal of Network Management, 27(6).

[For18] Forbes. (2018). 10 charts that will change your perspective of NetFlix’s massive success in the cloud. Retrieved from: https://www.forbes.com/sites/louiscolumbus/2018/07/12/10-charts-that-will-change-your-perspective-of-netflixs-massive-success-in-the-cloud/#6271cee62303

[Gei08] Geisberger R., Sanders P., Schultes D., & Delling D. (2008) Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In: McGeoch C.C. (eds) Experimental Algorithms. WEA 2008. Lecture Notes in Computer Science, vol 5038. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-68552-4_24

[Gei12] Geisberger R., Sanders P., Schultes D., & Vetter C. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. Transportation Science 46, 3 (08 2012), 388–404. DOI:https://doi.org/10.1287/trsc.1110.0401

[Git19] GitHub Inc. (2019). Create a learning switch. Retrieved from: https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch#ofp_match_class

[Ha17] Ha, T., Kim, J., & Nam, J. (2017). Design and deployment of low-delay hybrid cdn-p2p architecture for live video streaming over the web. Wireless Personal Communications: An International Journal, 94(3), 513-525. doi:10.1007/s11277-015-3144-1

[Har08] Hartenstein, H., & Laberteaux, K. (2008). A tutorial survey on vehicular ad hoc networks. Ieee Communications Magazine, 46(6), 164-171. doi:10.1109/MCOM.2008.4539481

[Jia17] Jia, Q., Xie, R., Huang, T., Liu, J., & Liu, Y. (2017). The collaboration for content delivery and network infrastructures: A survey. Ieee Access, 5. doi:10.1109/ACCESS.2017.2715824

[Kop07] Koponen, T., Chawla, M., Chun, B., Ermolinskiy, A., Kim, K., Shenker, S., & Stoica, I. (2007). A data-oriented (and beyond) network architecture. Acm Sigcomm Computer Communication Review, 37(4), 181-181. doi:10.1145/1282427.1282402

[Kri09] Krishnan R., Madhyastha H. V., Srinivasan S., Jain S., Krishnamurthy A., Anderson T., Gao J. (2009). Moving beyond end-to-end path information to optimize CDN performance. In Proceedings of the 9th ACM SIGCOMM conference on internet measurement, 190-201. Doi: 10.1145/1644893.1644917

[Kuz15] Kuzniar, M., Peresini, P., Kostic, D., & 16th International Conference on Passive and Active Measurement PAM 2015 16th International Conference on Passive and Active Measurement, PAM 2015 16 2015 03 19 - 2015 03 20. (2015). What you need to know about sdn flow tables. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),8995, 347-359. doi:10.1007/978-3-319-15509-8_26

[Kva09] Kvalbein, A., Hansen, A. F., Cicic, T., Gjessing, S., & Lysne, O. (2009). Multiple routing configurations for fast ip network recovery. Ieee/Acm Transactions on Networking, 17(2). https://doi.org/10.1109/TNET.2008.926507

[Lan10] Lantz, B., Heller, B., & Mckeown, N. (2010). A network in a laptop: Rapid prototyping for Software-Defined Networks. Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets 10. doi:10.1145/1868447.1868466

[Lan15] Lantz, B., & O'Connor, B. (2015). A mininet-based virtual testbed for distributed sdn development. Acm Sigcomm Computer Communication Review, 45(5), 365-366. doi:10.1145/2829988.2790030

[Mah18] Mahmoud, A., Abo Naser, A., Abu-Amara, M., Sheltami, T., & Nasser, N. (2018). Software-defined networking approach for enhanced evolved packet core network. International Journal of Communication Systems, 31(1). doi:10.1002/dac.3379

[Muk18] Mukund, B., & N, G. (2018). Route discovery for vehicular ad hoc networks using modified lion algorithm. Alexandria Engineering Journal, 57(4), 3075-3087. doi:10.1016/j.aej.2018.05.006

[Ntt11] Nippon Telegraph and Telephone Corporation. (2011). Welcome to RYU the Network Operating System(NOS). Retrieved from: https://ryu.readthedocs.io/en/latest/index.html

[Nyg10] Nygren, E., Sitaraman, R., & Sun, J. (2010). The akamai network: A platform for high-performance internet applications. Operating Systems Review (ACM), 44(3), 2-19. doi:10.1145/1842733.1842736

[Onf12] Open Networking Foundation. (2012). Software-Defined Networking: The New Norm for Networks.

[Onf19] Open Networking Foundation. (2019) SDN Overview. Retrieved from: https://www.opennetworking.org/sdn-definition/

[Sha18] Shamim, S., Badrul Alam Miah, M., & Islam, N. (2018). Data communication speed and network fault tolerant enhancement over software defined networking. Wireless Personal Communications: An International Journal, 101(4), 1807-1816. doi:10.1007/s11277-018-5759-5

[She05] Sherman, A., Lisiecki, P., Berkheimer, A., and Wein, J. (2005). ACMS: The Akamai configuration Management System. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, pp. 245-258.

[Swa19] Swami, R., Dave, M., V. (2019). Software-defined networking-based ddos defense mechanisms. ACM computing surveys, 52(2). Doi:10.1145/3301614

[Tan15] C. Tang, T. Kooburat, P. Venkatachalam, A. Chander, Z. Wen, A. Narayanan, P. Dowell, and R. Karl, “Holistic configuration management at Facebook,” in Proceedings of the 25th Symposium on Operating Systems Principles. ACM, 2015, pp.

328–343. [Online]. Available: http://doi.acm.org/10.1145/2815400.2815401

[Tei06] Teixeira, R., & Rexford, J. (2006). Managing routing disruptions in internet service provider networks. Ieee Communications Magazine, 44(3). https://doi.org/10.1109/MCOM.2006.1607880

[Unk15] Unknown. (2015). SDN OpenFlow. Using D-ITG Traffic Generator in Mininet. Demo tutorial Retrieved from: http://sdnopenflow.blogspot.com/2015/05/using-of-d-itg-traffic-generator-in.html

[Wan18] Wang, G., Liu, P., Zhao, Y., Li, J., & Song, M. (2018). Efficient openflow based inbound load balancing for enterprise networks. Procedia Computer Science, 129, 319-323. doi:10.1016/j.procs.2018.03.082

[Wen17] Wen, X., Bu, K., Yang, B., Chen, Y., Li, L., Chen, X., Leng, X. (2017). Rulescope: Inspecting forwarding faults for software-defined networking. Ieee/acm Transactions on Networking, 25(4). doi:10.1109/TNET.2017.2686443

[Wyt14] Wytrebowicz, J., Dinh, K. T., Kuklinski, S., & Ries, T. (2014). Sdn controller mechanisms for flexible and customized networking. International Journal of Electronics and Telecommunications, 60(4), 299–307. https://doi.org/10.2478/eletel-2014-0039

[Xia15] Xiao, X., & Kui, X. (2015). The characterizes of communication contacts between vehicles and intersections for software-defined vehicular networks. Mobile Networks and Applications: The Journal of Special Issues on Mobility of Systems, Users, Data and Computing, 20(1), 98-104. doi:10.1007/s11036-014-0535-6

[Xu17] Xu, Junbo & Rabinovich, Michael. (2017). NoCDN: scalable content delivery without a middleman. 1-6. 10.1145/3132465.3132476.

[Yan17] Yan, J., & Jin, D. (2017). A lightweight container-based virtual time system for software-defined network emulation. Journal of Simulation, 11(3), 253-266. doi:10.1057/s41273-016-0043-8

[Yaz16] Yazdani, M., & Jolai, F. (2016). Lion optimization algorithm (loa): A nature-inspired metaheuristic algorithm. Journal of Computational Design and Engineering, 3(1), 24-36. doi:10.1016/j.jcde.2015.06.003

[Yin19] Yin, X., Wang, L., & Jiang, S. (2019). A hierarchical mobility management scheme based on software defined networking. Peer-To-Peer Networking and Applications, 12(2), 310-325. doi:10.1007/s12083-017-0615-z

[Yu17] Yu, R., Xue, G., & Zhang, X. (2017). The critical network flow problem: migratability and survivability. Ieee/Acm Transactions on Networking, 25(6). https://doi.org/10.1109/TNET.2017.2747588