Genetic Algorithm-Based Improved Availability Approach for Controller Placement in SDN

by

Emmanuel Asamoah

Thesis submitted in partial fulfillment of the requirements for the
Master of Applied Science degree in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Thanks to the **S**oftware-**D**efined **N**etworking (SDN) paradigm, which segregates the control and data layers of traditional networks, large and scalable networks can now be dynamically configured and managed. It is a game-changing networking technology that provides increased flexibility and scalability through centralized management. The **C**ontroller **P**lacement **P**roblem (CPP), however, poses a crucial problem in SDN because it directly impacts the efficiency and performance of the network.

The CPP attempts to determine the most ideal number of controllers for any network and their corresponding relative positioning. This is to generally minimize communication delays between switches and controllers and maintain network reliability and resilience. In this thesis, we present a modified Genetic Algorithm (GA) technique to solve the CPP efficiently. Our approach makes use the GA's capabilities to obtain the best controller placement correlation based on important factors such as network delay, reliability and availability. We further optimize the process by means of certain deduced constraints to allow faster convergence.

In this study, our primary objective is to optimize the control plane design by identifying the optimal controller placement, which minimizes delay and significantly improves both the switch-to-controller and controller-to-controller link availability. We introduce an advanced genetic algorithm methodology and showcase a precise technique for optimizing the inherent availability constraints. To evaluate the trade-offs between the deployment of controllers and the associated costs of enhancing particular node link availabilities, we performed computational experiments on three distinct networks of varying sizes. Overall, our work contributes to the growth trajectory of SDN research by offering a novel GA-based resolution to the controller placement problem that can improve network performance and dependability.

Acknowledgements

I would like to begin by expressing my utmost gratitude to the Almighty God for His direction and guidance throughout my academic journey. I extend my heartfelt thanks to my supervisor, Professor Amiya Nayak, for his invaluable counsel and support during my thesis work. A very special thank you also goes out to Isaac Ampratwum, my dear brother and pastor, whose constant chaperoning, advice, and indispensable assistance propelled the completion of this research. Finally, I wish to express my profound gratitude to my family and friends for their unwavering support and inspiration as I conducted my research and wrote my thesis. Without them, this feat would not have been possible. I am thankful.

Contents

List of Tables

List of Figures

List of Acronyms

SDN Software-Defined Networking

CPP Controller Placement Problem

GA Genetic Algorithm

SC Switch to Controller

CC Controller to Controller

IP Internet Protocol

NFV Network Functions Virtualization

API Application Programming Interface

LP Linear Programming

ILP Integer Linear Programming

MILP Mixed Integer Linear Programming

NLP Nonlinear Programming

hINCEPT hybrid Incremental Controller Placement

hCPP hybrid Controller Placement Problem

RCP Reliable Controller Placement

RCP-DCP Reliable Controller Placement - Disjoint Control Path

RCP-DCR Reliable Controller Placement - Different Controller Replicas

WOA Whale Optimization Algorithm

VZ Virtual Zone

CH Cluster Head

VAN Virtual Application Network

ONOS Open Network Operating System

ITZ Internet Topology Zoo

UDP User Datagram Protocol

D-ITG Distributed Internet Traffic Generator

DCC Dynamic Controller Clustering

CCPP Capacitated Controller Placement Problem

DCCPP Dynamic Capacitated Controller Placement Problem

GRS Greedy Randomized Search

PSO Particle Swarm Optimization

CPP-FFA Controller Placement Problem Firefly Algorithm

POCO Pareto Optimal COntroller placement

PSA Pareto Simulated Annealing

MHNSGA Multi-Start Hybrid Non-Dominated Sorting Genetic Algorithm

CapEx Capital Expenditure

OPEX Operational Expenditure

MOCPP Multi-Objective Controller Placement Problem

PSO-CGLCPP Particle Swarm Optimization for Global Latency Control Placement Problem with Capacitated Controllers

MTTR Mean Time To Repair

MTTF Mean Time To Failure

SNDlib Survivable Network Design library

OEM Original Equipment Manufacturer

Chapter 1

Introduction

1.1 Motivation and Objective

Traditional networking infrastructure and its management approach have struggled to keep up with the ever-increasing complexity and dynamism of modern-day networks. Also, the ongoing advancement of information technology has enabled the Internet to build a complex infrastructure and massive foundation that has a significant impact on how people work and live.

Software-**D**efined **N**etworking (SDN) addresses these challenges by separating the control plane from the data plane, enabling centralized management and programmable control. However, this centralization introduces its own new concerns. The main prevalent one has to do with the problem of deciding on the optimal number and positioning of controllers to meet several pertinent standards such as ensuring efficient communication, minimizing delays and maintaining high availability. Furthermore, the trade-offs between these objectives must be cautiously considered, as they directly impact the network's performance, reliability, and resilience against failures. This **C**ontroller **P**lacement **P**roblem (CPP) is what we sought to, like other researchers, contribute to. It is quite

crucial in the SDN paradigm to achieve near ideal controller placement relative to certain specific or adaptive criteria. There are several of these criteria that serve as constraints necessary for optimum network operations. This has therefore proven to be a difficult research field and the CPP has been described as NP-hard [9]. Previous research efforts have attempted to address the CPP in SDN by considering isolated requirements. In contrast, our research and thesis approach takes into consideration some of the most critical factors of any network, including overall network delay and availability.

Availability refers to how often a system is operational or accessible, while reliability gauges consistent functionality and capacity to perform over time. Resilience also involves a network's ability to maintain service levels despite disruptions. While focusing on availability improvement, we have, however used these terms interchangeably in the sense that improving one often leads to improvements in the others. We treat the already complex topic of controller placement as a multi-objective optimization problem, taking into account the trade-offs between the various elements at play. By treating the CPP as a multi-objective problem, we aim to find solutions that optimize multiple objectives simultaneously, resulting in a more comprehensive and efficient network design

Stemming from how complex the controller placement problem is we decided to employ the **G**enetic **A**lgorithm which is a heuristic method that simulates natural selection and genetics. The GA is defined by the scholars in [10] as a stochastic search algorithm which functions on populations of possible solutions. This technique can explore multiple regions of any search space, handle complex and non-linear search spaces, and robustly handle noisy and sometimes incomplete data. Our algorithm is structured to essentially predict the most practical number and placement of controllers while still strictly adhering to switch to controller and controller to controller delay constraints. It further inculcates set threshold availability computations within the control plane as

well as between traffic forwarding nodes and their controllers. Our implementation of the genetic algorithm allows the ability to handle such multi-objective scenario efficiently and uncover the Pareto front of the problem [11].

1.2 Main Contribution

In this thesis, we use a GA-based framework to predict controller placement according to a number of requirements. The contributions of this thesis are as follows:

- We present a controller placement approach that considers delay constraints in the network. Our method aims to minimize the overall delay in the network while ensuring that each switch has access to at least two controllers for resilience.
- We determine highly available alternate routing between controllers, which further enhances network availability in case of controller failure.
- Lastly, we introduce a cost-based availability enhancement approach for controller-to-controller connections by adding node-disjoint recovery routes to restore paths. This approach ensures that the network can recover quickly in case of link or controller failure.

We run tests and simulations on three networks from the Survivable Network Design library[12] to assess the performance and effectiveness of our suggested strategy.

1.3 Thesis Organization

The remainder of the thesis is organized as follows.

- Chapter 2 reviews concepts of Software-Defined Networking, its architecture and existence as an emerging networking paradigm which has revolutionised traditional networks as we knew it. We then delve into some difficulties peculiar to SDN control plane and introduce the CPP and its general purpose.
- Chapter 3 reviews related work on CPP in SDN using different methods and algorithms.
- Chapter 4 introduces and discusses the Genetic Algorithm. We then describe our GA-based working model in detail and outline the flow of our approach and targeted results.
- Chapter 5 describes our data-set and the simulation setup. The experimental results are also presented and analyzed in this chapter.
- Chapter 6 concludes this thesis and proposes potential future works.

Chapter 2

Background

2.1 Software-Defined Networking

The internet has given rise to a digital culture in which practically everything is networked and accessible from anywhere. Despite their widespread acceptance, traditional IP networks are extremely complex and difficult to administer. It is challenging to configure networks based on predetermined policies and to reconfigure them to adapt to fault modifications, and load. The dilemma is worsened by the fact that such conventional networks are made up of nodes (i.e., switches, routers and so on) that act as a single entity, containing both the control plane (i.e., the brain) that makes decisions and the forwarding plane that forwards data packets [2]. Figure 2.1 shows an illustration of integrated management, control and data planes of a traditional switch. As a result, conventional networks have become overly complicated and unable to scale effectively.

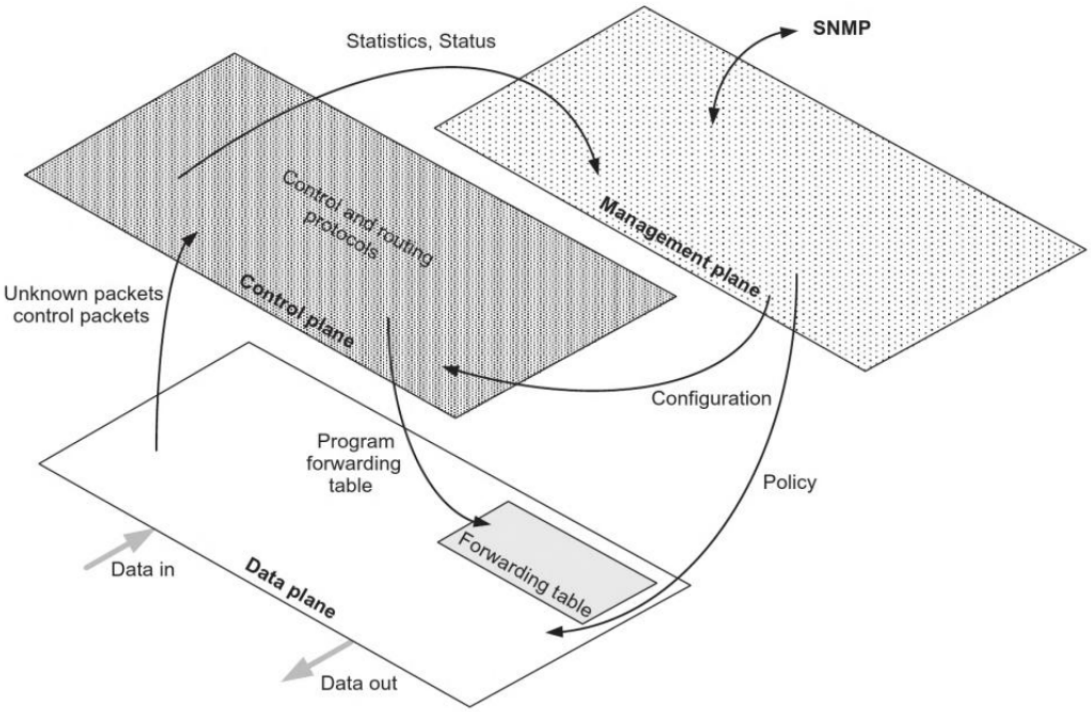


Figure 2.1: Role of the management, data and control planes - Single entity traditional switch [1].

The concept of SDN is believed to have originated from research efforts in the early 2000s. One of the pioneering papers on the subject, "A Clean Slate 4D Approach to Network Control and Management," was published by researchers at Stanford University in 2004 [13]. The paper proposed a novel architecture for network control that separated the control and data planes, providing a centralized control plane for managing network devices. SDN achieves this by decoupling the network node or switch into layers, abstracting the intelligence of the network into a centralized unit that controls network devices such as switches. This approach offers greater network flexibility compared to the limitations of traditional networks, enabling more efficient and effective management of network traffic. Figure 2.2 pins a typical traditional conventional network representation against SDN for pictorial comparison.

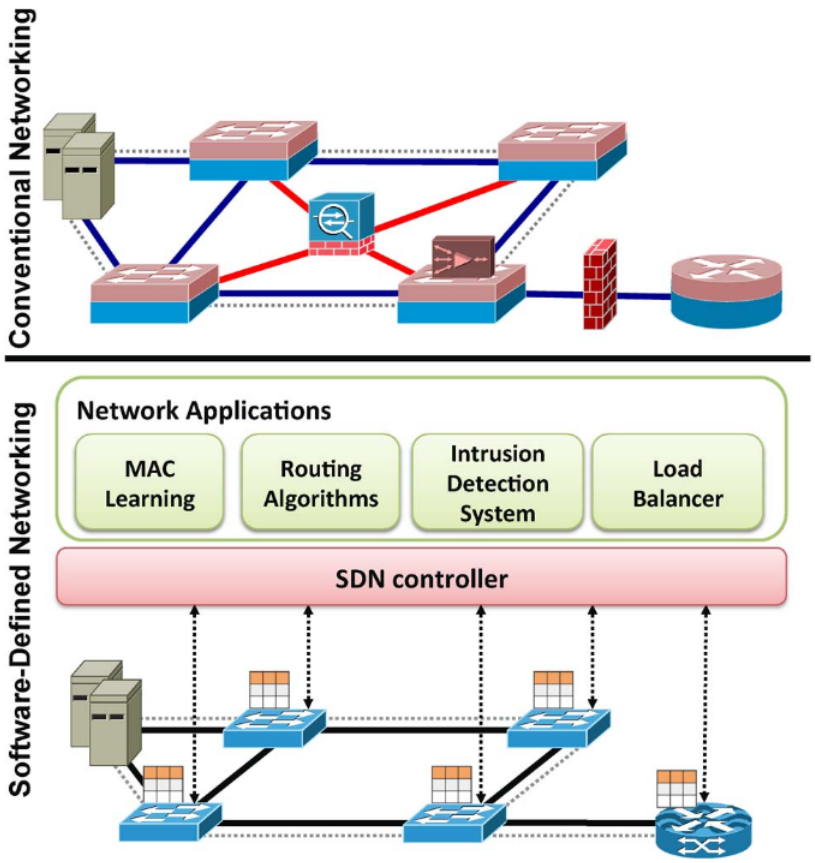


Figure 2.2: Traditional networking - switches and a router with integrated control and data plane **VRS** SDN - simpler, more manageable centralized control plane separate from the data forwarding boxes [2]

In the years that followed, SDN has continued to gain more traction as a preferred approach to network management. SDN is now widely adopted in a variety of industries, including telecommunications, cloud computing, and enterprise networks. SDN has enabled new network topologies and applications, such as network function virtualization (NFV) and network slicing, which offer new methods of managing and deploying network services.

2.1.1 SDN Architecture

The SDN Architecture consists of three primary planes: the application plane, control plane, and forwarding or data plane [14]. Each of these planes serves a distinct function.

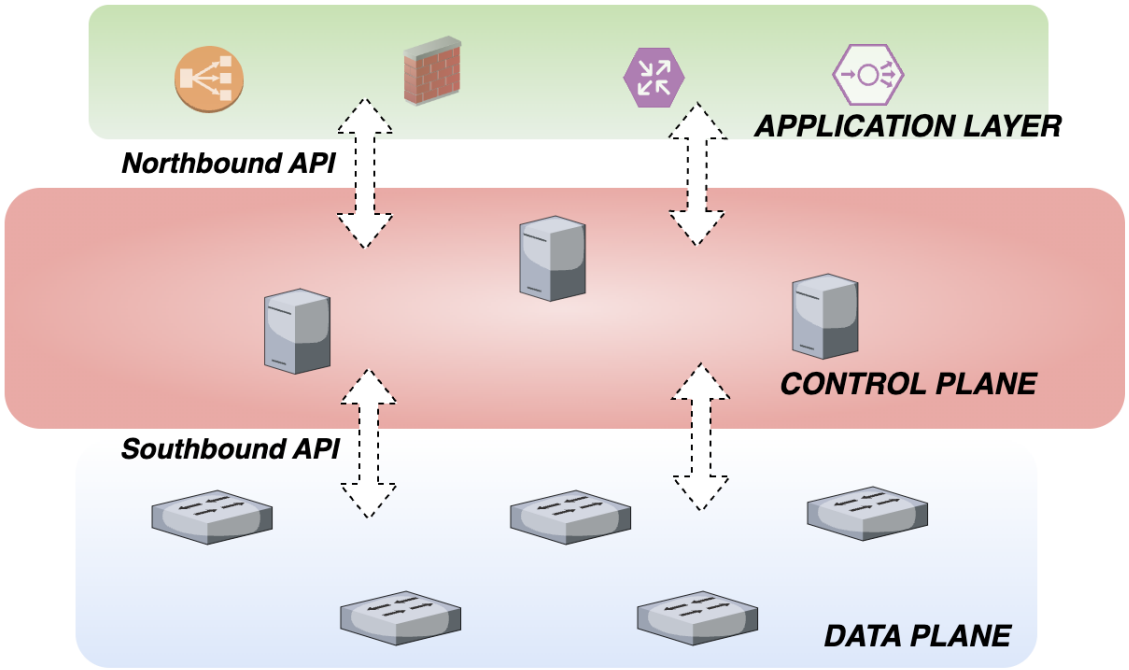


Figure 2.3: Basic SDN Architecture

The data plane is in charge of packet buffering, scheduling and overall data forwarding. The application layer houses applications with specific capabilities that are deployed within the SDN infrastructure. These programs include capabilities like traffic engineering, load balancing, and security [15]. Network administrators interact through such apps with the SDN structure via management. The SDN control plane serves as the central component of the SDN architecture. It provides a centralized control plane for managing and configuring network devices in accordance with network policies and needs. The controller communicates with network devices and offers network managers with a

programmatic interface for defining network policies and configuring network devices. Keeping the forwarding table always up to date is the significant purpose of the control plane. It is responsible for control and routing protocol processing that directly affects the forwarding table of the data plane [1]. The central controller communicates with data plane nodes/switches via southbound APIs and with application plane components by means of northbound APIs. Figure 2.3 is a representation of the SDN architecture.

2.1.2 SDN Challenges

SDN has presented various advantages to network administration, such as centralized control and efficient resource usage [16]. However, there are still challenges associated with the implementation of SDN.

A major challenge in SDN is network security. As network architecture evolves and expands, new security vulnerabilities may emerge, posing potential network threats. Because the centralized controller provides a single point of attack, SDN security vulnerabilities may occur as a result of the expanded attack surface [17]. Furthermore, network device programmability and the adoption of open-source software may create new security vulnerabilities.

Another significant difficulty with SDN is the issue of single point of failure. Since SDN architecture relies on centralization, the controller serves as the brain of the network. If the controller fails, the entire network goes down, resulting in serious disruption and potentially substantial financial losses. Moreover, the centralized nature of the controller makes it an attractive target for attackers [18]. A successful attack on the controller could result in the entire network being compromised or taken offline.

While SDN has demonstrated superior scalability in comparison to traditional network architectures, it has also introduced its own unique set of challenges and limitations.

As the network size grows, a single controller may no longer be sufficient to effectively manage all the network responsibilities. To address this, network administrators may need to add more controllers as the network expands. The architecture can be implemented with a physically or logically centralized or distributed control plane [19]. However, adding more controllers raises the question of where to place them essentially. The placement of controllers is a critical issue, as it impacts the overall performance and efficiency of the network. This is popularly known as the **"Controller Placement Problem" (CPP)**, and it involves determining the optimal quantity and placement of controllers in the network to ensure efficient and effective management.

2.2 Controller Placement Problem

Controller Placement Problem aims to discover ideal positions of SDN controllers in order to meet several set goals such as latency reduction, load balancing, energy saving, and increased reliability. It is a combinatorial optimization challenge. The best location of controllers is often determined by various criteria, including network architecture, traffic load, and network device arrangement [20]. The difficulty is exacerbated by the fact that the appropriate placement of controllers is frequently context-dependent and may vary depending on the network's specific requirements [21].

Over time, multiple approaches to the CPP have been proposed and developed. We look at some of these in Chapter 3. Each approach employs its own algorithm(s) and has its own respective strengths and weaknesses considering several factors; some of which are discussed below:

Cost is an important factor within the context of SDN, the terms Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) are taken into account when minimizing

the overall cost associated with controller placement [22]. Obtaining positive results for minimal expense is achieved by considering a wide range of costs, such as budgetary constraints, procurement, repair, and maintenance costs [23].

Latency represents a critical consideration in CPP. The determination of optimal controller placement is heavily influenced by the distance between network nodes, which plays a significant role in the propagation of data packets. The optimality of latencies has a direct impact on both Controller-to-Controller (CC) and Switch-to-Controller (SC) communication delays [24].

Fault tolerance inculcates a level of redundancy to ensure consistent switch to controller communication and network management. Failure of network connectivity can have a severe negative impact on controlled switches and may limit control plane functionality. This is especially true when the connection between the switch and the controller is lost owing to a failed link or a faulty controller. The switch will no longer receive updated routing instructions, resulting in dropped packets [25]. Thus, it is quite necessary to consider controller placement that prioritizes fault tolerance and redundancy.

Inter-Controller Communication plays an important role in achieving global consistency in the context of the CPP by maintaining inter-controller communication through state synchronization. Controllers communicate with each other to send signals to switches controlled by other controllers, affecting the performance of end-to-end communication between switches managed by different controllers. Minimized inter-controller delay greatly impact this positively and should be factored in controller placement design.

Generally, as previously mentioned, CPP is highly context-specific and may vary significantly depending on the specific requirements of the network. In some cases, it may

be necessary to combine various performance metrics to address this complex problem as a multi-objective optimization challenge [26]. While this approach further complicates the problem, it ultimately enables the identification of an optimal solution that best meets all of the required criteria. This is the precise approach we adopted in our thesis as shown in Section 4.2.

Chapter 3

Related Work

In the quest to advance research in this field and attempt to provide feasible solutions to the CPP with multiple controllers in SDN, various approaches, methods, and algorithms have been employed. These techniques are broadly categorized with some criteria overlapping or purposefully integrated. In this chapter, we discuss, explore and elaborate on some of these well-known methods.

3.1 Linear Programming and Mathematical Approaches

Linear programming (LP) is a mathematical optimization method that has been widely used to solve linear problems that are subject to constraints. LP can be used to model the CPP in SDN as a collection of linear equations and inequalities, which can then be used to find the best solution by minimizing or maximizing a linear objective function. LP is a good fit for linear problems and has been shown to work effectively in certain instances of the CPP. Simplex and interior-point methods are two examples of LP algorithms.

Despite its ability to provide exact answers, LP can be computationally costly when

dealing with large-scale scenarios, which is a significant disadvantage [27]. As the network grows in size, so does the number of variables and constraints, resulting in a more complicated optimization issue that necessitates more computer resources to solve. Furthermore, LP may not be appropriate for non-linear or non-convex optimization situations. In that regard, researchers have developed heuristic algorithms, as shown in Section 3.3, among other techniques to generate near-optimal solutions toward the CPP in SDN in a more computationally efficient way.

Other mathematical approaches such as Integer Programming (IP), Mixed-Integer Linear Programming (MILP), and Nonlinear Programming (NLP) have also been used to model and solve the CPP in SDN. These techniques enable precise formulations of the problem and can yield exact solutions. Their significant downside, however, is that they can be computationally complex, which makes them less suitable for large-scale problems.

3.1.1 Optimal and Expansion Model for the CPP in SDN

The research team in [27] commenced by focusing solely on the cost of controllers and offered an LP model to tackle the CPP in SDN. The suggested CPP approach sought to reduce controller deployment costs while determining the best number of controllers, locations, types, links with varying processing capacity, and bandwidths. The scheme was tested through simulations, and the results demonstrated that it was fitting for small-scale SDNs. The approach, however, was found to be inefficient in terms of time and space, as 10% of the problem could not be addressed within the requisite time of 30 hours, and memory limits were reached. Furthermore, the proposed strategy was created for cloud-based networks for a variety of businesses.

The limitations identified highlighted the need for more efficient and scalable algo-

rithms for solving the CPP in SDN. It reiterated the necessity in considering not only the cost of controllers but also other factors such as network delay, availability, and resilience to make the placement of controllers more effective. However, the same team, in [28] tried to minimize update costs in case of network expansion or addition of extra switches. Linear Programming (LP) was again formulated using the network design and the list of extra switches to search for a network reorganization that minimizes the update cost. The simulations conducted showed the effectiveness of the proposed approach in planning a new network or updating an existing one. Moreover, introducing another controller required five links, and the computational time was approximately 4.60 ms.

3.1.2 hINCEPT for hCPP

INcremental **C**ontoll**E**r **P**lacemen**T** for hybrid SDN and legacy networks. The **hINCEPT** was proposed by [3] as an efficient hybrid CPP method. The proposed approach aimed primarily to provide network operators with a migration path and to maximize switch to controller control channel resilience by lowering the failure probability of the SDN switch's control channel throughout the legacy network to SDN migration planning stages. See Figure 3.1.

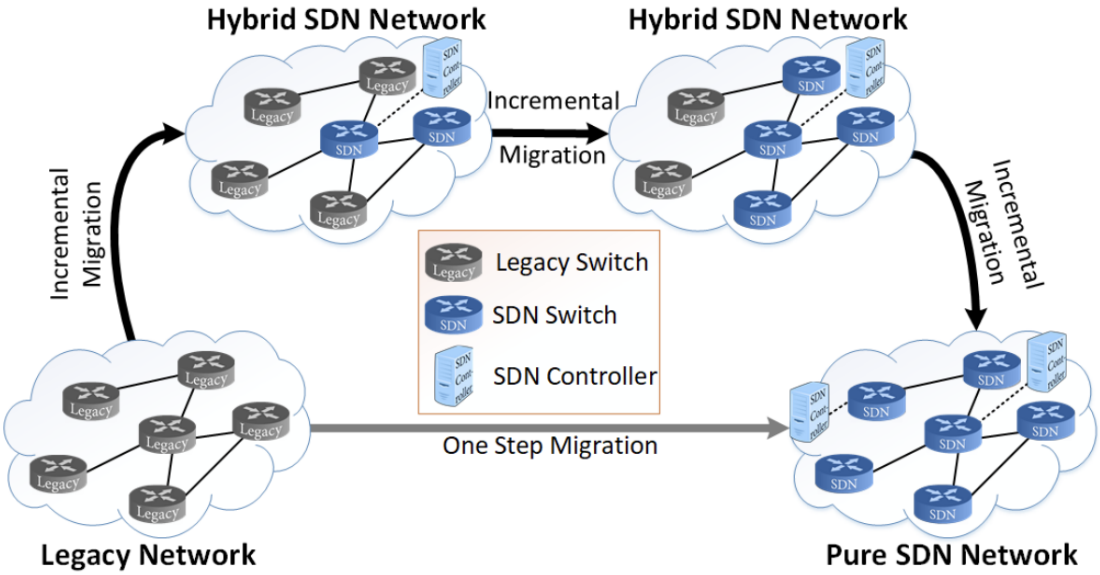


Figure 3.1: SDN Migration and controller placement: one-step migration **VS.** incremental migration [3]

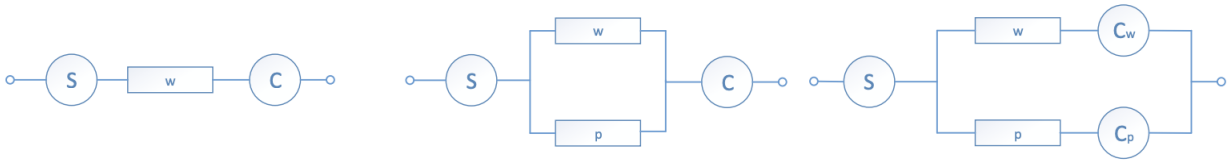
The hCPP approach improves proper decision making in deciding legacy nodes replacement with SDN switches, when, where, and how many controllers that can augment the resiliency. Using an Integer Linear Programming (ILP) technique, an optimization problem was constructed over time. The number of controllers, channel resiliency, and controller utilization were the performance indicators studied. The hCPP approach was evaluated against other existing approaches, and it was found to achieve a considerably higher level of resilience regardless of the number of controllers.

The results showed that hCPP achieved about 77% higher resiliency with about 33% fewer controllers, and a consequence of this was a 200% utilization. This shows that the hCPP approach is an efficient method for hybrid SDN and legacy networks in achieving high resiliency while reducing the number of controllers and increasing the utilization of the network resources.

3.1.3 RCP in SDN

Also, we discuss the concept of **R**eliable **C**ontroller **P**lacement(RCP) in SDN proposed by researchers in [4]. They presented this as an approach to improve control plane availability and protect it against single link and node failures, hence providing continuous fail-over backup control pathways utilizing resilient routing principles.

The researchers applied Mixed Integer Linear Programming (MILP) and presented two strategies. The first method highlighted switch to controller (SC) delay and assumed that switches must be connected to a controller through two Disjoint Control Paths. This was referred to as RCP-disjoint control path (RCP-DCP). The second method demanded that switches must be connected to two Different Controller Replicas (RCP-DCR) through two separate or disjoint pathways. The two techniques are shown in Figure 3.2 together with a representation of an unprotected control path



(a) Unprotected control path $A = a_s a_w a_c$

(b) Disjoint Control Paths (RCP-DCP) $A = a_s(1-(1-a_w)(1-a_p))a_c$

(c) Different Controller Replicas (RCP-DCR) $A = a_s(1-(1-a_w a_{c_w})(1-a_p a_{c_p}))$

Figure 3.2: Reliability block diagram and control path availability expressions for **Unprotected** , **RCP-DCP** and **RCP-DCR** models. S stands for switch; w for working and p for protection control path; C for controller (C_w: working and C_p: backup controller) [4]

The RCP-DCP and RCP-DCR approaches were designed to enable fast and efficient failover with minimal impact. Simulations were performed on real network topologies, they revealed considerable improvement in the resilience of the control plane for both approaches, but with an extra cost to the average control path with about a 2% increase in length for most topologies. In the case of link failures, both RCP-DCP and RCP-DCR

delivered the same performance with an applicable strategy determined by topological features, characteristics and number of controllers. However, for node failures, RCP-DCP performed better, protecting controllers against failures.

3.2 Clustering Approaches

The clustering approach is a reliable, practical technique for taking on the controller placement problem in SDN. By partitioning the network into smaller, more manageable regions dependent on criteria such as geographical proximity or network topology, this approach offers a scalable solution for large and complex networks. The controller placement problem is solved independently within each region, ensuring efficient allocation of controllers to minimize latency, balance load or achieve whichever set criteria specific to the network.

Inter-region connectivity is established to maintain overall network performance and seamless communication between different network segments [29]. This method also allows for adaptation and reconfiguration to account for changes in traffic patterns or network topology, ensuring that the controller placement remains optimal over time.

The benefits of the clustering approach are numerous. Its ability to scale and adapt enables it to better handle sizable networks, while its ability to distribute controllers across the network enhances fault tolerance and a high level of resilience in that it ensures continued network operation in the event of controller failures. Furthermore, load balancing is achieved by distributing the workload among multiple controllers, preventing any single controller from becoming a performance bottleneck and ensuring a more balanced and efficient network operation.

Additionally, the clustering approach is versatile, allowing it to be customized to spe-

cific network requirements, such as latency constraints, traffic patterns, or geographical distribution. This adaptability ensures more precise and effective controller placement, ultimately leading to improved network performance and resource utilization. We look at varied attempts at CPP in SDN via clustering.

3.2.1 W.O.A SDN Clustering

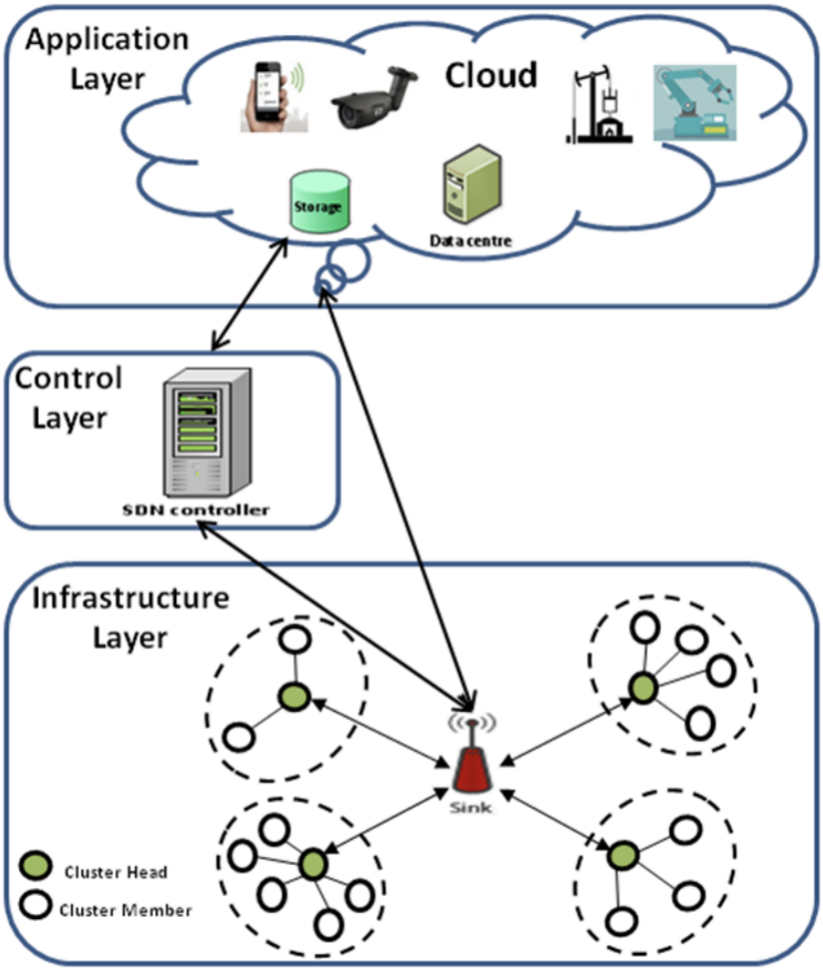


Figure 3.3: W.O.A-based SDN Architecture [5]

The Whale Optimization Algorithm (WOA) was proposed as a sophisticated and innovative method that addresses both sensor resource constraints and the varied distribution of node density across a geographical area. The proposed protocol aims to enhance network performance and optimize resource utilization by carefully considering multiple factors in the clustering process of deciding network controllers and their placements [5].

The methodology of this technique initially divides the sensing area into virtual zones (VZs), which serve as a foundation for achieving a balanced distribution of cluster heads (CHs) according to the node density within each VZ. This partitioning method helps accommodate the uneven distribution of nodes throughout the network, allowing for more efficient resource allocation and improved network management. As displayed in Figure 3.3, the protocol functions within an architecture that provides proper segregation of the control plane and the data plane utilising the concept of SDN.

W.O.A commences by establishing a random set of solution options for each virtual zone (VZ); each outcome refers to the Identification Address (ID) of a particular group of cluster heads (CHs). The search agents (solutions) then upgrade their positions at each iteration, following either a randomly chosen search agent (solution) or the previously obtained optimal solution. Furthermore, the solutions are assessed, and the expense or

Table 3.1: WOA parameters

α	Energy parameter
β	Distance parameter
τ	Density parameter

cost of the following function is minimized:

$$cost = \alpha f_1 + \beta f_2 + \tau f_3 \tag{i}$$

$$f_1 = \frac{\sum_{i=1}^{SG} E_{(n_i)}}{\sum_{j=1}^{K} E_{(CH_{(p,j)})}} \tag{ii}$$

$$f_2 = \sum_{j=1}^{K} \frac{\sum_{i=1}^{NC_{p,j}} dis_{(n_i,CH_{p,j})}}{NC_{p,j}} + \frac{\sum_{j=1}^{K} dis_{(CH_{p,j},BS)}}{K} \tag{iii}$$

$$f_3 = \frac{SG}{\sum_{j=1}^{K} CMs_{(j,CH_j)}} \tag{iv}$$

According to [5], K denotes the quantity of cluster heads (CHs). Other parameters defined in Table 3.1 which are network specific are the energy, distance and density traits. In (ii), the function f_1 selects the collection of cluster heads with the highest energy value. The transmission cost between cluster heads and cluster members, as well as the cluster members and the sink, is included in the function f_2 in (iii). Furthermore, the node density is processed in (iv), which selects the collection of cluster heads that covers the greatest number of nodes. In the cluster heads selection procedure, the SDN controller uses the information contained in the nodes that are stored in the cloud. The process is summarized categorically in the following phases:

- **Division phase:** This entails the sensing area being segmented or divided into virtual zones **(VZs)** and the nodes defined into sub-groups **(SGs)**.

- **Initialisation phase:** This is where random vectors of search agents are initialised (sets of cluster heads IDs) and parameters specific to the network are agreed on as well as the criteria to terminate the algorithm or maximum iteration value.

- **Evaluation phase:** The search agents are evaluated at this stage according to function f_1 in (i) and the current best search agents(set of cluster heads) are selected and assigned as optimal solutions agent.

- **Update phase:** the update criteria are utilized in exploitation and exploration of the search vectors while still limiting search agent with regards to the cluster heads IDs.

- The evaluation and updates phases continue repetitively until the termination criteria or maximum iteration value is exhausted.

- The whole process is replicated for each virtual zone to produce optimal results.

Following the creation of VZs, the Whale Optimization Algorithm is employed to select the optimal set of CHs. The WOA is a **bio-inspired** optimization algorithm that mimics the social behavior of humpback whales. It considers crucial factors such as residual energy, communication cost, and node density to make informed decisions on CH selection [30]. By incorporating these factors into the optimization process, the proposed approach ensures that the selected CHs contribute to a more energy-efficient, cost-effective, and well-balanced network. The use of W.O.A in clustering for SDN brings forth a dynamic and adaptable network that can respond effectively to changes in node densities and resource availability. This adaptive quality is essential for maintaining robust and efficient network operation, as it enables the network to continually optimize CH distribution and resource management.

3.2.2 Clustered Distributed Controller Architecture

The novel clustered distributed controller architecture for SDNs in the paper by [6] addresses scalability, fault tolerance, and interoperability challenges in centralized control. By utilizing multiple SDN controllers working together in a cluster, this architecture efficiently handles connected OpenFlow switches and performs load balancing. As a result, the network can accommodate expansion or elasticity, remain resilient in the face of controller defects, and ensure seamless integration with existing deployments through standard SDN protocols such as OpenFlow.

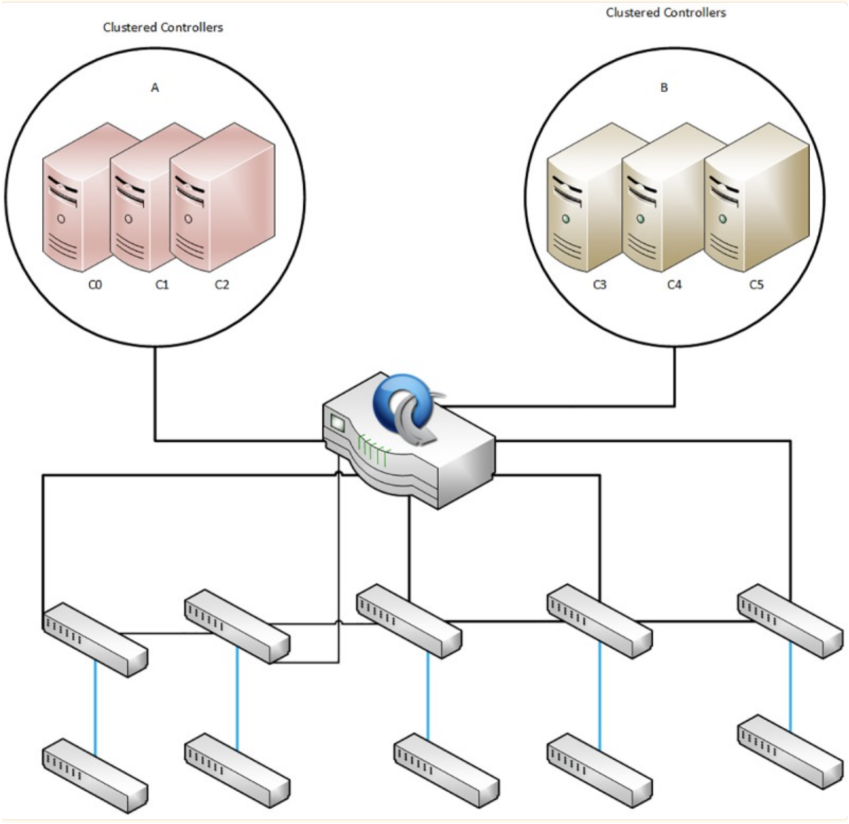


Figure 3.4: Suggested distributed controller clustering in [6]

The representation in Figure 3.4 depicts clustering **A** that showcases the implementation of a Virtual Application Network(VAN) SDN controller installed across three

separate Amazon EC2 servers, each running the 64-bit Ubuntu Server Edition 14.0 LTS. The network topology and environment are specifically designed to adhere to HP's stipulations, which dictate that there should be no loops among OpenFlow switches and that all switches must be under the control of the controller. The VAN SDN controller can be set up in two different modes: Standalone mode and Team mode. The proposed method employs the Team mode, which offers high availability and automatic fail-over. This ensures that the network remains managed continuously, even if one of the team controllers becomes non-operational.

In clustering **B**, we see the Open Network Operating System (ONOS) controller deployed on three individual servers, each running the Amazon EC2 Ubuntu Server 64-bit 14.0 LTS edition. The Rest API is utilized to cluster multiple controllers, enabling them to share data, maintain network consistency state, and manage OpenFlow switches. Each switch is connected to the primary controller, with IP addresses indicating the standby controllers for every connected switch. The ONOS controllers are configured in an equal mode (Active-Active) within the cluster. As a result, the clustered controllers achieve load balancing by distributing the connected OpenFlow switches among the various controller instances in the cluster.

Two tests were conducted to assess the proposed architecture. The first one on latency was done using, among others, a standard topology from the Internet Topology Zoo (ITZ) with 25 nodes(switches), 25 hosts and 30 connected links. They made use of the controller bench marker application for the test and recorded positive results. The second experiment was done to capture the number of dropped packets in the instance of a controller fail-over. This was done by continuously streaming UDP packets between two end host devices using the Distributed Internet traffic generator (D-ITG). The test results generally provided less latency and drops relative to networks without the proposed

architecture.

Essentially, their study introduced a distributed controller clustering approach in SDN to address the single point of failure issue while also tackling state distribution, data sharing, and consistency challenges in distributed SDN architectures. As illustrated in Figure 3.4, the proposed system's architecture comprises multiple controllers organized into a cluster, with each controller acting as a "team member" and one designated as the team manager (Primary Controller). The clustering configuration, initiated on one controller, automatically propagates to other team members, irrespective of which controller assumes the team manager role. The primary controller oversees configuration and monitoring of controllers and switches, and in case of failure, the highest-priority controller within the cluster takes over as the primary controller. The recovered failed controller resumes operations as a regular team member. The proposed technique's controller clustering configuration takes into account a minimum cluster size of three controllers, identical controller versions across the cluster, individual IP addresses for each controller, and a dedicated IP address assigned to the cluster itself.

3.2.3 2-phase Dynamic Controller Clustering algorithm

The paper [31] introduces a innovative approach to address the CPP and its load balancing problem in the SDN control plane, reducing the load on the central element while maintaining the maximum distance constraint between controllers. A two-tiered load balancing architecture was proposed, featuring a Super Controller at the top level that organizes controllers into clusters for balanced load distribution. At the lower level, a dedicated Master Controller in each cluster reassigns switches to balance loads among controllers. A two-phase Dynamic Controllers Clustering algorithm is provided for top-level load balancing, executed at regular intervals with adjustable cycle lengths to allow

for more accurate load balancing results. The top-level operation can run independently of the bottom-level operation, enabling shorter cycle times. Theoretical analysis suggests that the algorithm offers a near-optimal solution, and simulation results indicate a five-fold improvement in dynamic clustering over fixed clustering. The technique is simplified into procedures and heuristics which compliment each other for its full functionality.

Algorithm 1: 2-phase DCC Algorithm [31]

Input: nt Network contain $C = \{c_1, c_2, \ldots, c_M\}$ Controller list, and distances between controllers. K and M for the number of clusters and controllers, respectively. *constaintActive* to indicate that it meets the controller-to-controller distance constraint. $offset$ to calculate the distance constraint (optional).

Output: $P = \{p_1, p_2, \ldots, p_k\}$ Clusters list, where $P_i = \{c_i1, c_i2, \ldots, c_i(m/k)\}$

```
procedure: if constraintActive = true then
    Masters ← Algorithm 1(nt);
    (initialDistanceClusters, maxDistance) ← Heuristic 1(C, Masters);
    Cnt ← maxDistance + offset;
    finalPartition ← Procedure 1(initialDistanceClusters, true, Cnt);
    else
    initialStructure ← Cluster structure from the previous cycle;
    initialLoadsOnly ← Heuristic 2(c);
    initialWithReplacement ← Procedure 1(initialLoadsOnly, false);
    ReplacementOnly ← Procedure 1(initialStructure, false);
    finalPartition ← best solution from(initialLoadsOnly,
    initialWithReplacement, ReplacementOnly);
    end if
    return finalPartition;
```

The Dynamic Controllers Clustering (DCC) Algorithm was designed to create clusters of controllers within a Software-Defined Network (SDN) control plane and begins by considering the network topology, distances between controllers, and an optional distance constraint. The algorithm receives a network "nt" containing a list of controllers "C", the distances between them, and the number of clusters "K" and controllers "M". Additionally, it takes a Boolean flag "*constraintActive*" to indicate if the distance constraint between controllers should be considered or not, as well as an optional "$offset$"

to adjust the distance constraint. The output of the algorithm is a list of clusters "P", where each cluster "P_i" contains a subset of controllers.

The algorithm's procedure starts by checking if the distance constraint is active, meaning if "*constraintActive*" is set to true. If it is, the algorithm first calls Algorithm 1 with the network "nt" to obtain the master controllers ("$Masters$"). Next, it calls the distance initialization process (Heuristic 1) with the controller list "C" and the "$Masters$" to obtain the initial distance clusters and the maximum distance between controllers. Afterward, it calculates "Cnt" by adding the "$offset$" to the maximum distance. Finally, it calls the initial distance clusters, the true flag, and "Cnt" to create the final partition of clusters.

On the other hand, if the "*constraintActive*" is set to false, the algorithm uses the cluster structure from the previous cycle as the initial structure. It then calls the second heuristic with the controller list "C" to calculate the initial loads only. Following this, Procedure 1 is called with the initial loads only and the false flag to generate the initial partition with replacement. After that, Procedure 1 is called again, but this time with the initial structure and the false flag to generate the replacement-only partition. Lastly, the algorithm chooses the best solution among the three generated partitions (initial loads only, initial with replacement, replacement only) based on minimal load differences.

3.3 Heuristic Approach

The heuristic approach entails leveraging practical guidelines to quickly find approximate solutions, particularly when dealing with complicated or large-scale issues. It is a problem-solving technique that employs shortcuts in a limited time frame to create nearly perfect solutions. [32]

The greedy algorithm, which chooses the most favorable option at each stage; simulated annealing, which steers clear of local optima by exploring better and worse solutions; tabu search, which prevents cycling through already seen solutions; genetic algorithms, inspired by natural selection, which evolve a population of candidate solutions; and ant colony optimization, which uses artificial ants to cooperatively find solutions, are some common heuristic algorithms. We explore some relevant works in finding CPP in SDN using heuristics based algorithms in this section and go further to propose our genetic algorithm based model in Section 4.2.

3.3.1 Dynamic Capacitated CPP in 5G based on SDN/NFV Architecture

In the paper by [33] they suggested a heuristic multi-objective optimisation method using a Dynamic Capacitated Controller Placement (DCCPP) algorithm. This was implemented on a distributed SDN network integrated with Network Function Virtualization for 5G-CN It was built on the basis of the K-center problem in order to solve the capacitated controller placement problem (CCPP), which functioned as a resource location problem in which the location and number of controllers were allocated in order to optimize resources. For the purpose of scheduling, rescheduling and accomplishing load balancing, a Greedy Randomized Search (GRS) algorithm was used to solve the dynamic assignment of nodes to controllers.

3.3.2 CPP-PSO

The paper published in [34] used Particle Swarm Optimization (PSO) and Firefly algorithm, two population-based meta-heuristic algorithms for optimal controller placement that take a specific set of objective functions and return the best possible location

from them. The CPP was formulated with controller to switch latency as well as inter-controller latency in consideration. The firefly algorithm, also referred to as the CPP-FFA outperformed the PSO when tested on the Networks from the TopologyZoo website. The FFA was designed based on features identical to a firely flickering.

The researchers in [35] offered a multi-critical method comparison of solution techniques using different swarm optimization algorithms. They focus on different swarm intelligence methods which inculcate factors including financial and capital expenditures (CAPEX) and operating costs (OPEX) for the installation of networks and maintenance. Moreover, PSO is used to respond to the CPP controller placements. The techniques used determine the approximate optimal number of controllers needed for servicing an SDN, as well as their locations and switch distribution.

3.3.3 Pareto-based Optimal COntroller placement - POCO

POCO conducts an exhaustive assessment of all possible candidate placements in its default configuration. While this is doable for small and medium-sized networks, realistic time and resource constraints necessitate an alternative approach in the context of large scale networks or dynamic networks with changing properties [36]. The POCO toolset was thus supplemented with a less accurate but faster heuristic. Using a measure to quantify the error introduced by the heuristic method allowed for analysis of the resulting trade-off between time and accuracy. Aside its usefulness in determining feasible CPP solutions, POCO also facilitates visualizing the related solution space in a graphical user interface. This is accessible and was implemented as open source software [11].

They designed an algorithm with structured input which was divided into two sections. There was data unique to the problem, such as the topology graph G and the desired number of controllers k. Secondly, the Pareto Simulated Annealing (PSA) mech-

anism had parameters such as placement evaluation number per iteration, number of iterations per temperature level as well as the annealing scheduling control features [37]. The evaluation of the implemented PSA revolved mainly around quantifying the disparity between the amount of time saved by using a heuristic method and the accuracy loss.

3.3.4 Multi-Start Hybrid Non-Dominated Sorting GA - MHNSGA

A multi-objective controller placement approach that uses a fast and efficient adaptation of evolutionary algorithms for large-scale SDNs was suggested by [21]. Their system utilizes a heuristic algorithm called Multi-Start Hybrid Non-Dominated Sorting Genetic Algorithm (MHNSGA). The objective metrics they considered included SC-latency, CC-latency, load balancing, and reliability for link failures. The algorithm is targeted at computing the Pareto Optimal Control Placement (POCO) using the Pareto front in the objective space and the Pareto set in the decision space [38]. The algorithm starts with a high-quality initial population and uses a fast Pareto finder to efficiently search for the Pareto optimal front.

The assessment of MHNSGA was performed on several network topologies and compared with other approaches such as POCO, Pareto Simulated Annealing (PSA), and PSO-CGLCPP in terms of time and search space. Results uncovered that MHNSGA outperformed the other approaches in terms of computation time and space, with an average deviation of about 0.8% in comparison with the original Pareto optimal set. The scheme has the capability of exploring a large portion of the search space and generates accurate approximations of the Pareto optimal front about 20 times faster than the other approaches.The results showed that MHNSGA is efficient in the context of solving Multi-Objective Controller Placement Problem (MOCPP).

3.4 Summary

This chapter, in a nutshell, reviews various methods leveraged in the attempt to provide solutions to the CPP in SDN. The approaches looked at spans across mathematical programming, clustering techniques and heuristics algorithms. Certain strategies, such as the Whale Optimization Algorithm, referenced in 3.2.1, epitomize a hybrid technique, embodying aspects of both clustering and heuristic methodologies.

We have highlighted the targets, strengths and drawbacks throughout the assessment and review of each approach.

Chapter 4

The Genetic Algorithm and the Proposed GA-based CPP Optimization Model

4.1 The Genetic Algorithm

4.1.1 Introduction

Genetic Algorithms (GAs) are now widely used in engineering teaching and learning as an adaptive method for solving complicated issues and problems. The technique is believed to have been first introduced by John Holland in the early 1970s. It is a meta-heuristic method for resolving challenges in mixed computation and solving optimization problems [39]. GA manages the finding system strategy with the help of the **selection**, **crossover**, and **mutation** operators. This algorithm is inspired by natural selection and genetic principles and pertains to the broad category of evolutionary algorithms used in computing and informatics [8].

Algorithm 2: Pseudo-Code of a basic Genetic Algorithm

Require: Data, Pop (population size), Iteration (number of iterations)
Ensure: Best Solution

```
Generate Pop
while not Finish() do
    for i = 1 to Pop do
        Keep the best solution
        Evaluate Objective Function(i)
    end for
    Fitness Function()
    Apply GA operators Pop(t) → Pop(t + 1)
    Evaluate Pop(t + 1)
end while
```

In GAs, potential solutions to a problem are represented as chromosomes, which are usually encoded as strings of characters or binary digits. Each individual in the population symbolizes a potential solution to the issue, and the population is made up of a set of these chromosomes.

As shown in Algorithm 2 the population is thought of as a method of generation for each reproduction in evolution, which typically begins with a community of randomly selected people. The population's overall fitness is assessed for each generation. This is typically the objective feature being sought after or solved for as the best solution for the environment [40]. Based on a randomised probability, individuals that are deemed adequately fit within the existing population are chosen for their genes to be modified to birth a new generation cycle for all (recombined and potentially mutated at random) [41]. Over the next generation of the process, newer candidate tactics would be applied. It adheres to the concept of "survival of the fittest": Improved solutions emerge from earlier generations until a nearly ideal solution is attained. When the algorithm has generated a maximal number of generations or is satisfied, it typically comes to an end. In order to add to the search in a region of the improved outcome within a coverage

framework, GA represents an intelligent use of random search supported with historical data. A simplified flowchart of a GA is depicted in Figure 4.1.

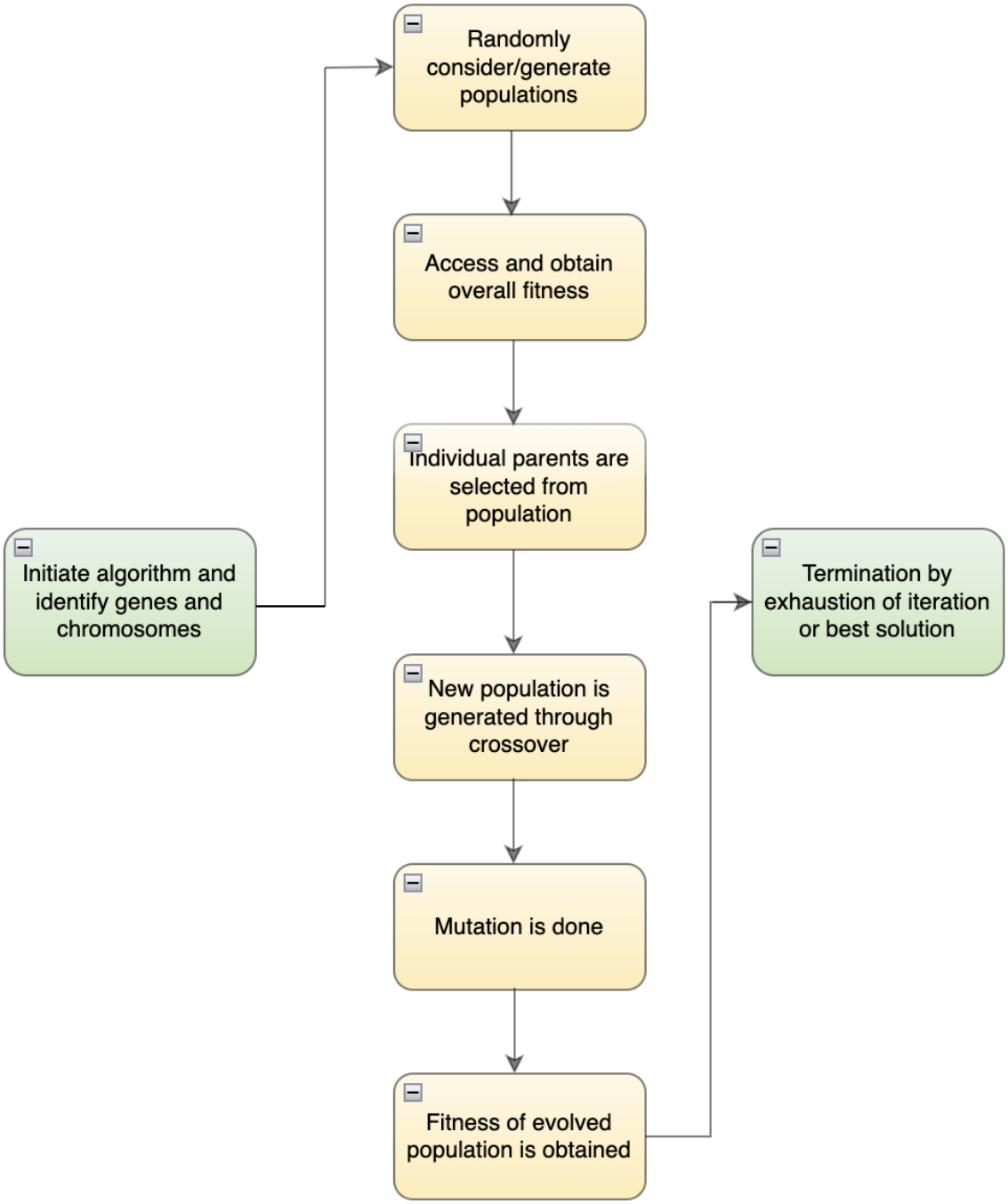


Figure 4.1: Genetic Algorithm

4.1.2 Genetic Algorithm Methodology

4.1.2.1 Initialization

The first stage in a GA involves initializing the population. This entails the development of a collection of feasible answers to the issue, commonly referred to as individual genes or chromosomes. The initial population is frequently generated randomly, and each chromosome symbolizes a potential solution to the optimization issue. The variety of solutions produced by this random generation covers the entire search area [8]. The complexity of the issue determines the population size, which is usually chosen to strike a balance between exploration and exploitation of the search space.

Chromosomes are encoded representations of possible solutions, and the success of the algorithm depends greatly on choosing the right representation. Chromosomes can be expressed as binary strings, integer strings, floating-point numbers, or other domain-specific structures based on the problem formation [42]. It is crucial to pick a representation that reflects the characteristics of the issue while enabling effective manipulation and evaluation of the solutions.

4.1.2.2 Selection

At the time of each succeeding generation, a percentage of the existing populace is selected to breed a new generation process. The individual answer or solution is selected in order to use the health feature using a fitness-based process. The fundamental methods of assessment evaluates each outcome's health to determine the best course of action. As this method can be very time-consuming, other methods also score a representative sample of the populations [43]. The majority of functions are stochastic and created so that only a small percentage of less accurate solutions are chosen. This aids in preserving high levels of community heterogeneity and prevents excessive convergence on subpar

responses.

4.1.2.3 Reproductions

The next stage is to create a second generation populations of solutions from those chosen through crossover (also known as recombination) and/or mutation, two genetic operators. Pairs of "parent" solutions are handpicked from the pool that was previously chosen to breed in order to create novel offspring solutions. By using the aforementioned crossover and mutation techniques to create a "child" solution, a new solution is produced that frequently exhibits or bears traits from both of its "parent" solutions. Every new child may become a new parent, and this cycle will repeat itself until the population has grown to the anticipated numbers. The populace of the new generation is created with various chromosomes. Therefore, this process raises the populations' average level of fitness.

Crossover Operators: The main objective is to produce offspring by fusing the genetic makeup of the parent chromosomes, encouraging the interchange of advantageous traits and accelerating convergence towards ideal outcomes. Two common crossover techniques include single-point crossover, which involves selecting a random crossover point and exchanging genetic material to the left (or right) of this point, and multi-point crossover, where multiple crossover points are chosen, and the genetic material between these points is alternately swapped between parents, encouraging a higher degree of genetic exchange. Depicted in Figure 4.2 is a single-point and two-point crossover.

Mutation Operators: The primary goal is to introduce small, random changes into an individual's chromosome in order to preserve population diversity and avoid premature convergence to less-than-ideal solutions. This is shown in Figure 4.3. Mutation encourages exploration of the search space and prevents the algorithm from getting trapped in

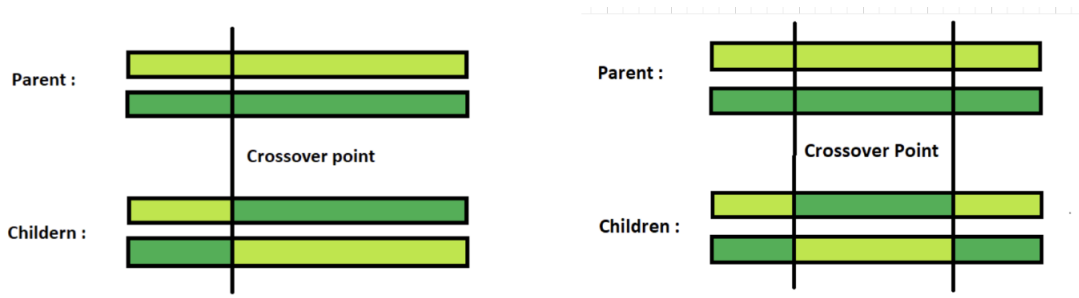


Figure 4.2: Single-point and two-point Crossover [7]

local optima by sporadically perturbing the genetic material. This operator is essential in striking a balance between exploration and exploitation, which eventually improves the algorithm's capacity to solve challenging optimization problems at their best.

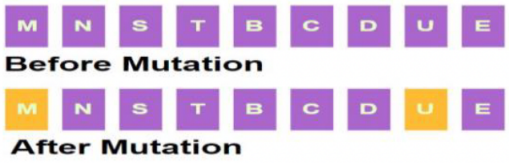


Figure 4.3: Mutation [8]

Even though methods of reproduction based on the use of two parents are more "biology inspired", some study indicates it may be preferable to use more than two "parents" in order to reproduce a high-quality chromosome [44]. These processes eventually lead to a population of chromosomes in the subsequent generation that differs from the first generation. Since only the best genetic algorithm traits from a preceding generation are chosen for breeding, along with a small percentage of less fit solutions, for the reasons already stated above, the population's overall average fitness will have increased as a result of this process.

4.1.2.4 Termination

The genetic algorithms generational process is repeated or iterated until a termination condition has been reached. Common terminating conditions are:

- Finding a solution that meets the minimum requirement criteria;
- The highest ranking solution's fitness is approaching or has hit a plateau, meaning that further iterations no longer yield better results;
- The allocated budget of resources (computation time/money) has been exhausted or reached;
- Manual inspection;
- A combination of the above.

4.1.3 Applications Areas of the Genetic Algorithms

4.1.3.1 Machine Learning

In genetics-based machine learning, which is an emerging field, the GA is used. The adaptability of GAs allows for their application in various machine learning paradigms, including supervised, unsupervised, and reinforcement learning. Moreover, GAs can be employed in both shallow and deep learning architectures, enhancing the performance of traditional machine learning algorithms and deep neural networks alike. There are some reasons why GAs are important to machine learning. First off, gradient-based techniques are not applicable in discrete spaces where GAs operations thrive [10]. This might be utilized to look for rulesets, neural network structures, cellular automation devices, and many other things. This could be used in this manner, while stochastic high scaling and optimization methods could also be considered.

Also, they serve as useful methods for reinforcement learning. Fitness is a specific variable used to evaluate the effectiveness of the learning method. GA requires a population, and often what is required is not a solitary individual, but a community. Training in multi-agent structures is a prime illustration of this [45].

4.1.3.2 Economics

In economics and financial systems, genetic algorithms (GAs) are being utilised to solve complicated optimization problems including the cobweb system, portfolio optimization, the creation of trading strategies, market simulations, and game-theoretic issues. Additionally, they support risk management, option pricing, credit rating, and financial forecasting. The ability of GAs to adapt makes them suitable for the various challenges in these areas, allowing for effective search space exploration and exploitation. Numerous economists have begun to employ GAs to address typical flaws in traditional economic models while preserving an accessible mathematical framework. GA is optimized in a manner that can represent the complexity of actual problem-solving situations. Additionally, it does not require a fixed structure for issues regarding policy, allowing it to choose between different optimizations [46].

4.1.3.3 Robotics Formation and Control Systems

Genetic Algorithms (GAs) have found widespread application in robotics and control systems, addressing complex optimization problems in fields such as path planning, motion control, inverse kinematics, and robot design. They also help with swarm robotics, gait optimization, robot learning, sensor fusion, fault tolerance, and human-robot interface [47]. The adaptability and global search capabilities of GAs make them ideal for solving a wide range of robotics problems. The GA can be used to design the path that

a robotic arm takes from one step to the next. Over the last few generations, various GA implementations in the field of robotic trajectory modeling have been carried out. This versatile public-purpose optimization algorithm has been used to generate collision-free paths for a robot with specified begin and goal mutual specifications. The primary features of the algorithm's design are a code system and fitness analysis. The fitness assessment includes a total distance, time, and collision repercussions for each route.

4.1.3.4 Optimization Problems

The GA has been most frequently leveraged in optimization problems where researchers must maximize or minimize an objective function value under a particular set of constraints or parameters. Such a plan will address an optimization-related issue. The optimization raises questions about how to minimize or optimize variables that have a number of factors that are usually subject to restrictions on fairness and inequalities. This is not only crucial in research purposes, but also to operations in study, marketing, and manufacturing [41]. Several industrial engineering architecture problems are extremely complex and difficult to solve with conventional optimization algorithms. Due to its potential as a cutting-edge optimization methodology, GA has garnered a great deal of notice in recent years. GAs typically excel at resolving optimization issues because of their capacity to conduct a global search, adaptability to various domains, inherent parallelism, and resistance to local optima. The resolution of disputes between objectives to produce a set of Pareto-optimal solutions is another trait that GAs are adept at [41]. In other words, in terms of optimization, the GA is capable of achieving a solution that cannot be dominated by any another with regards to all the objectives no matter how conflicting they may be. These attributes make GAs a flexible option for a variety of optimization tasks.

4.2 Proposed Enhanced GA-based Model for CPP in SDN

The CPP is an undoubtedly critical aspect of SDN network design. The optimal placement of controllers is primarily constrained by delays between nodes (switches) and their corresponding controllers as well as among the controllers themselves. Several other factors come into play specific to certain networks or specified criteria.

The concept of **availability**, in conjunction with overall network delay, however, has not been an area of focus by most researchers. Availability generally refers to the probability or likelihood that a specific path or group of paths between nodes in a network remains operational and accessible during a given time period. It is an essential aspect of network performance and reliability, as it directly impacts the overall functioning and quality of experience for users in the network [48]. Inter-controller path availability as well as control paths from nodes(switches) to their corresponding controllers are two very salient factors we inculcate in our work.

The CPP is considered and known to be NP-hard [49]. This makes the problem a very complex one to which we propose a heuristic technique to determine near accurate predictions of controller placement. Our research tackles this complex optimization problem by directly handling and addressing the following issues:

(A) controller placement designed to meet delay constraints;

(B) extraction of a sub-graph establishing shortest links between all nodes in the controller set via the Steiner tree concept to improve availability;

(C) determination of highly available alternate routing between controllers;

(D) assigning each node to two controllers for redundancy;

(E) cost of availability enhancement for controller-to-controller connections by adding node-disjoint recovery routes (restore paths).

This problem is inherently multi-objective. We base our methodology on the genetic algorithm approach to solve the problem above.

4.2.1 CPP Initial Problem Formulation

The SDN data plane is modeled as a graph $G = (N, E)$, where N represents the set of nodes (switches), and E represents the set of links. Each link is denoted by its end nodes $\{i, j\}$. The delay between two nodes, represented by d_{ij}, is assumed to be proportional to the shortest path length between them [50]. There are two delay constraints discussed in this scenario:

(i) **Switch-to-controller delay ($D_{\mathbf{sc}}$)**: The delay between each switch and the controller that manages it must not exceed a given maximum value D_{sc}. This constraint is necessary to ensure efficient communication and control between the switches and their corresponding controllers.

(ii) **Controller-to-controller delay ($D_{\mathbf{cc}}$)**: The delay between any two controllers must not exceed a given maximum value D_{cc}. This constraint ensures that the controllers can communicate effectively with each other for coordination and synchronisation purposes.

It is assumed that the communication between controllers and the switches they manage occurs more frequently than the inter-controller communication. Therefore, D_{sc} is expected to be smaller than D_{cc} ($D_{\text{sc}} < D_{\text{cc}}$) [51]. This implies that the placement of controllers should prioritize minimizing the switch-to-controller delay while still maintaining an acceptable controller-to-controller delay.

4.2.2 Inter-controller Primary Path Sub-graph - Steiner Tree

Given a controller placement solution, a possible sub-graph is deduced creating a link of shortest routes between all controllers in real time. Similar to [52], we represent our sub-network with a Steiner graph and assume its terminal nodes are the controllers. The Steiner tree is used to connect multiple SDN controllers, ensuring that the primary paths between them fall on the least cost links with the aim of satisfying the maximum inter-controller delay value D_{cc}. Utilizing the Steiner tree allows for efficient routing and minimizes the overall communication cost between controllers in the network.

To further increase inter-controller communication availability , our proposed model determines feasible supplementary protection routes among the controllers and allows highly available controller backup paths to be created. In the event that the primary paths fail or experience issues or cuts, having backup paths guarantees that communication between controllers continues. A node-disjoint backup route for any two controllers is computed. This means that the backup path has no nodes in common with the primary path (except for the source and destination controllers) and is not forced to use the Steiner tree.

The network is better prepared to manage failures or performance issues by implementing both primary and node-disjoint backup paths. This approach improves the SDN control plane's overall reliability and fault tolerance, guaranteeing uninterrupted communication between controllers even in adverse circumstances. It is assumed that each network link availability value depends only on the link distance. A link's availability is an indicator of its reliability, demonstrating the likelihood that the link will be operational at any particular moment [53]. Distance-based availability suggests that a link's reliability is directly proportional to the physical distance it covers.

The availability, A_{ij}, of a link in a network between nodes i and j can be calculated

using the following formula:

$$A_{ij} = 1 - \frac{MTTR}{MTTF_{ij}} \tag{4.1}$$

where,

- $MTTR$ (Mean Time To Repair) is the average time required to repair a failed link which is calculated in hours;
- $MTTF_{ij}$ (Mean Time To Failure) is the average time to failure, also in hours. It represents the mean period between failures of any link $\{i, j\} \in E$ before the link fails.

For the purpose of our tests, we assume $MTTR = 24hrs$. $MTTF_{ij}$ however, is given by:

$$MTTF_{ij} = \frac{CC \times 365 \times 24}{l_{ij}} \tag{4.2}$$

where CC, the cable cut rate is taken as 450km and the link distance is l_{ij}.

This formula gives the availability of the link as a value between 0 and 1 (or 0% and 100%). A higher value indicates a more reliable link, while a lower value suggests that the link experiences more frequent failures or longer repair times.

Our objective goal is to ensure that the end-to-end availability between any two controllers is at least a given value K, which is five nines **(0.99999)**. To accomplish this, we compute and evaluate additional restore paths for any particular controller-to-controller connection if the availability of the primary and backup paths is **less** than five nines [54]. One key approach to enhance end-to-end availability is through path redundancy.

Given primary and backup paths with availability A and B connecting controllers $C1$ to $C2$, the effective path availability of $C1 - C2$ is calculated as $1 - (1 - A)(1 - B)$. If the computed availability is less than our required five nines, we then go ahead to find N node-disjoint restore routes that can improve the availability of $C1 - C2$ to five nines.

By doing this, our proposed model identifies and incorporates additional redundant paths that are node-disjoint, ensuring that the overall availability between any two controllers meets or exceeds the desired threshold of five nines. By increasing path redundancy and strategically selecting node-disjoint restore routes, we can enhance the end-to-end availability and ensure the robustness of the network.

4.2.3 Switch to Controller Connection

For any given switch, our model primarily targets finding and assigning to the nearest controller, which implies meeting the maximum delay D_{sc} constraint. Additionally, our algorithm has a secondary goal of ensuring an average of four nines **(0.9999)** for all switch-controller connections.

To accomplish this, we identify an additional controller for each switch, aside from its primary controller, that is within a range that will yield a switch-to-controller availability of an average of four nines in the network. Figure 4.4 shows a simple reference node (switch) S, and its assigned controller $C1$ with availability A_1. Our proposed model creates a node-disjoint connection setup to an auxiliary controller $C2$ with availability A_2 also, such that the association of the switch to the control plane possesses an effective four nines availability $\mathbf{1 - (1-}A_1\mathbf{)(1-}A_2\mathbf{) = 0.9999}$.

Four nines up-time equates to a maximum of 52 minutes of outage per year [55]. By selecting an additional controller within the specified range, we can enhance the reliability and availability of the switch-controller connections. This approach ensures that even

if the primary controller fails, the switch can maintain its connection with a secondary controller, maintaining the desired level of availability in the network. By optimizing switch-controller connections and prioritizing the nearest controller while also ensuring a secondary controller within the defined range, our algorithm aims to improve overall network performance and resilience.

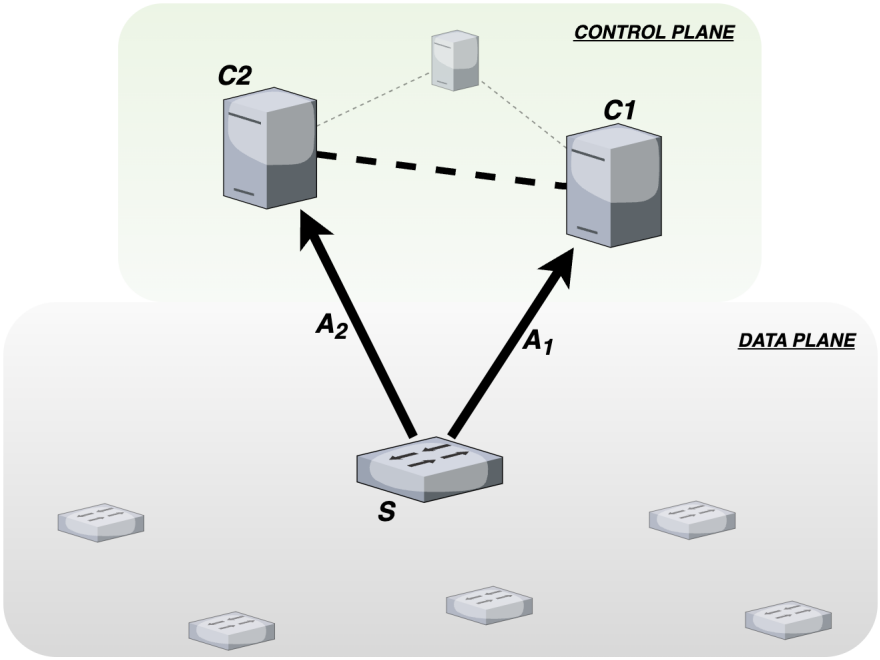


Figure 4.4: S-C connections

4.2.4 Methodolgy

We leverage the GA to solve the optimization problem that represents the CPP in SDN. In the context of the CPP, our GA begins by initializing a population of candidate controller placement solutions, represented as chromosomes. Each chromosome encodes the number and positions of the controllers in the network topology. The fitness function evaluates the quality of each solution, based on metrics defined.

The GA iteratively refines the population by applying selection, crossover, and mutation operators. The selection process favors fitter solutions, while the crossover and mutation operators generate new offspring solutions by combining and perturbing the genetic material of the parent chromosomes.

By exploring and exploiting the search space, the GA converges towards an optimal or near-optimal solution for the CPP. The final solution provides the best trade-off between the objectives specified. Our Pareto frontier essentially now revolves around minimizing latency or delay constraints and maximizing controller and node availability, and can be used to guide the deployment of controllers in real-world SDN networks. Our flowchart is displayed in Figure 4.5. We outline below the sequence of steps in our algorithm.

(1) Our algorithm starts by going through the various possible controller count we can have in the network. For example, in a network with 5 nodes, we consider the possibility of 1 up to 5 controllers. The optimization goal however still remains, that is, identifying the minimum number of controllers needed to fulfill the outlined constraints. We initialize a list of solutions accordingly.

(2) For given number of controllers, we generate the initial population.

(3) For each solution candidate, we check if the individual satisfies the constraints. As described in Section 4.2.1 we consider an SDN data plane which is represented by a graph $G = (N, E)$, where N, E stands for the set of nodes and set of links respectively. $\{i, j\}$ represent link end nodes and we assume the delay between two nodes is proportionate to the shortest route length between them and is denoted as d_{ij} .

Switch-Controller delay cannot exceed D_{sc} while inter-controller delay limit is D_{cc}. We postulate that the maximum delay between a switch and its managing controller

must not surpass the specified maximum threshold delay of inter-controller communication $D_{\text{sc}} < D_{\text{cc}}$, given that the former occurs more frequently than the latter.

The constraints are as follows:

- for any node, there should be 2 controllers, one distanced at most D_{sc} from it as shown in (4.3). This is the primary controller that a node is connected to. A secondary controller is introduced for node-controller reliability. The second controller should be at a distance such that the availability between the node and any of the two controllers is at least four nines as explained in Section 4.2.3 - (4.4).

$$\sum_{j \in N} y_j \geq 2 \quad \forall i \in N : d_{ij} \leq D_{sc} \tag{4.3}$$

$$1 - (1 - A_{ij_{primary}})(1 - A_{ij_{secondary}}) \geq \mathbf{0.9999} \tag{4.4}$$

- any two controllers y_i and y_j cannot be positioned further than D_{cc} apart. Nodes placed beyond the established threshold are precluded from concurrently serving as controllers - (4.5).

$$y_i + y_j \leq 1 \quad \forall i, j \in N : d_{ij} > D_{cc} \tag{4.5}$$

(4) Calculate the fitness of each individual. The fitness is detailed in Section 4.2.4.2.

(5) Update GA population through selection, crossover and mutation as specified in Section 4.2.4.3.

(6) Repeat Step 2 through to 4 until the GA stopping criteria is met.

(7) Add the solution derived for considered number of controllers to solutions list.

(8) Repeat Step 2 through to 7 for all possible numbers of controllers.

(9) Choose the minimum set of controllers with the minimum fitness value satisfying all the considered constraints.

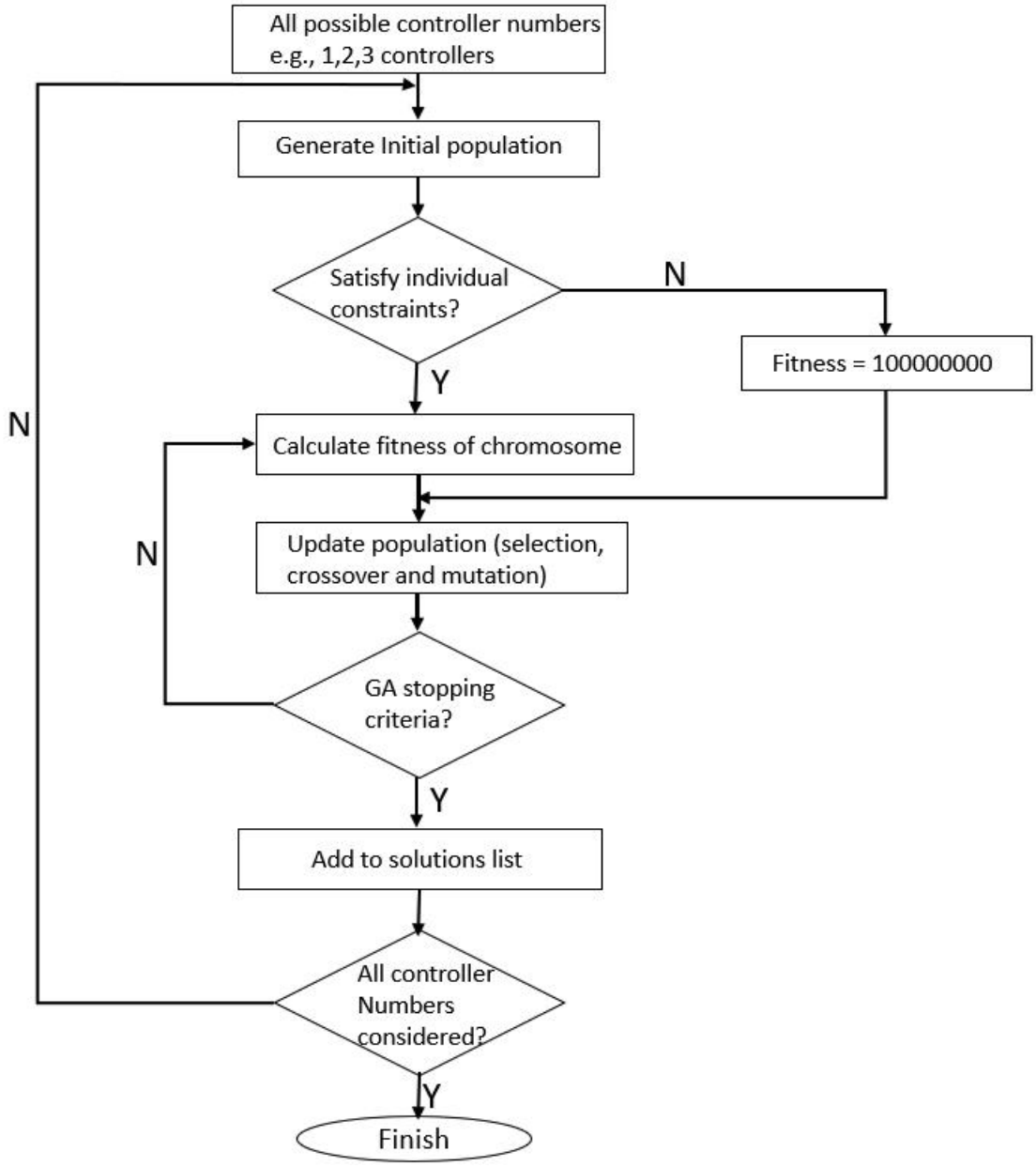


Figure 4.5: GA flowchart

4.2.4.1 Initialization and Population Generation

The first step of our proposed method takes into account the generation of a population to constitute the controllers for the given network. The goal is to determine which network node locations should be chosen to host controllers. Each chromosome in the initial population corresponds to a valid controller placement scheme. This is accomplished through the use of a binary coded genetic method. Each chromosome represents one method of controller selection and is represented as a binary array whose length equals the total number of nodes in the network (each bit corresponds to one network node). Each bit in the array with a value of 1 indicates that the associated node has been designated as a controller, while zeros indicate regular nodes. The initial population is randomly generated and the fitness of each chromosome is computed.

4.2.4.2 Fitness Function

Following the determining of the controller locations, each regular node is allocated to the controller that is in closest proximity to it. If the controller placement satisfy the solution constraints discussed above, we compute the fitness of the individual. The goal is to achieve a controller placement solution such that the problem constraints are met at a minimized cost. These costs involve two distinct components:

- First, the quantity of supplementary restore paths needed to attain a five nines availability for all controller to controller connections;
- And second, the average delay incurred in both switch-to-controller(S-C) and controller-to-controller (C-C) communications via primary paths. This comprehensive approach aims to provide an efficient and robust solution to controller placement in the context of network optimization.

$$\text{Fitness Function} = \min(\alpha \times N_{\text{restore paths}} + \beta \times \bar{D})$$

i Fitness Function is the objective function we want to minimize.

ii α and β are weighting factors that determine the relative importance of the number of restore paths and the average delay in the optimization. $\alpha = 1000$; $\beta = 1$. We assign a higher value to α because restore paths consume network link resources and the goal is the avoid that as much as possible.

iii $N_{\text{restore paths}}$ is the number of restore paths required to achieve a certain level of availability for controller to controller connections.

iv $\bar{D}$ is the average delay for communication between network nodes (S-C and C-C).

4.2.4.3 Selection, Crossover, Mutation and Termination

Individuals are chosen or selected using the Roulette-Wheel technique [41], with the probability of being chosen increasing with the fitness value of the individual chromosome. We used the single-point crossover to create new offspring from the selected parents.

Figure 4.6 depicts an example of single-point crossover, where k = 1 (also known as single point crossover), with two parent chromosomes, Parent A and Parent B, in a network with 10 nodes showing 4 placed controllers according to the fitness of the previous generated population. Following the crossing, two new child chromosomes are formed, Offspring X and Offspring Y.

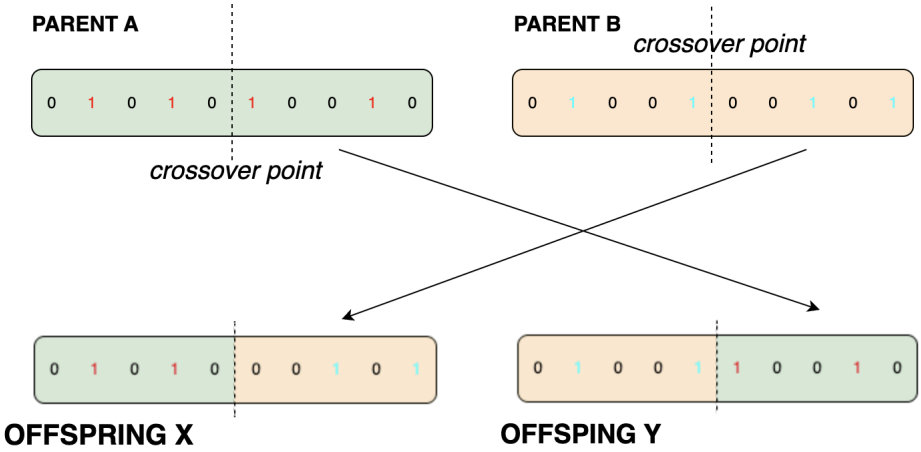


Figure 4.6: Single-point crossover

Afterwards, mutation is carried out to augment the offspring. To perform the mutation over an individual, we assigned a mutation probability to every gene in the chromosome. Each gene probability is then compared with the mutation rate. If the mutation probability of a gene is less than the rate, we swap the gene. The number of controller positions in a chromosome is always controlled to match the controller numbers being considered for the iteration. We use a mutation probability of 0.1.

Termination condition of genetic algorithm can be defined as as either reaching a predefined number of iterations or not observing further improvement in the population. Our termination condition is set at 100 iterations.

4.2.4.4 Parameter Sensitivity Analysis

In our research endeavor, we investigated best practices pertaining to the recombination parameters of the Genetic Algorithm (GA) and endeavored to fine-tune them in a form of a grid search specific to our requirements.

Mutation Rate : Our investigation divulged that an inflated mutation rate might induce the GA to exhibit behavior resembling a random search, whereas an excessively low mutation rate may not infuse sufficient diversity into the algorithm. To ascertain the optimal balance, we kept other parameters constant and systematically evaluated the correlation between various mutation rates and their subsequent impact on switch-to-controller (S-C) availability. This empirical approach as shown in Figure 4.7 revealed an abrupt ascension in availability when the mutation rate was increased from 0.01 to approximately 0.05, following which the availability plateaued. Upon conducting an exhaustive series of experiments, we discovered that a mutation rate of 0.1 engendered the most efficacious and pragmatic results.

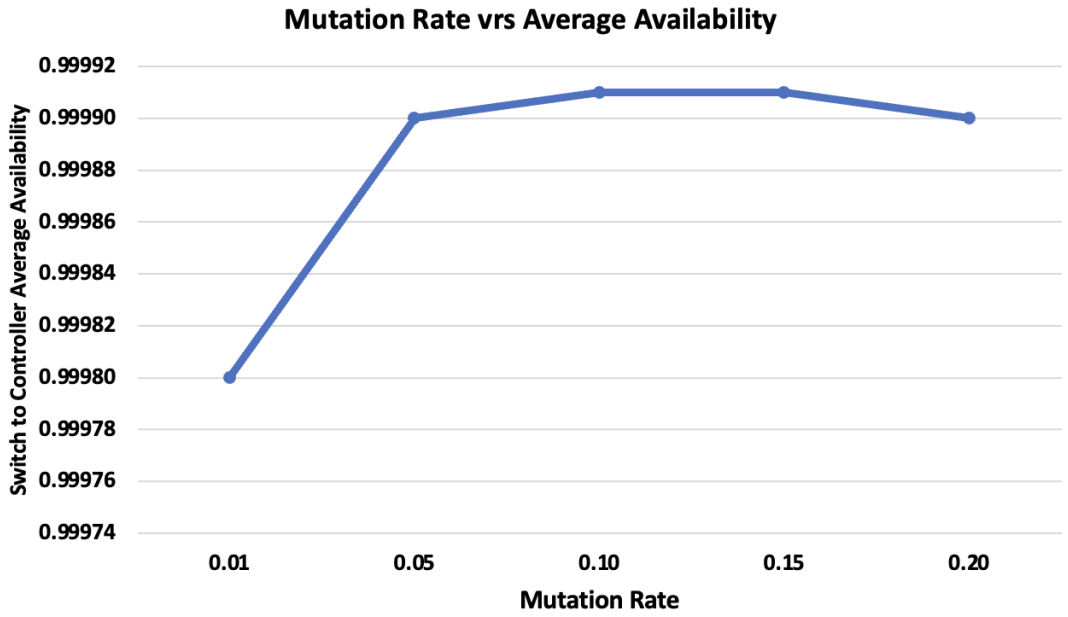


Figure 4.7: Mutation probability = 0.10

Termination: In the early stages of running our algorithm, we noticed a swift rise in the average availability from the switch to the controller as the number of iterations

increased. However, a saturation point was observed at the 100-iteration mark. Beyond this point, spending more time running additional iterations didn't lead to any significant improvements in target availability. Hence, continuing with a higher number of iterations seemed inefficient and unnecessary.

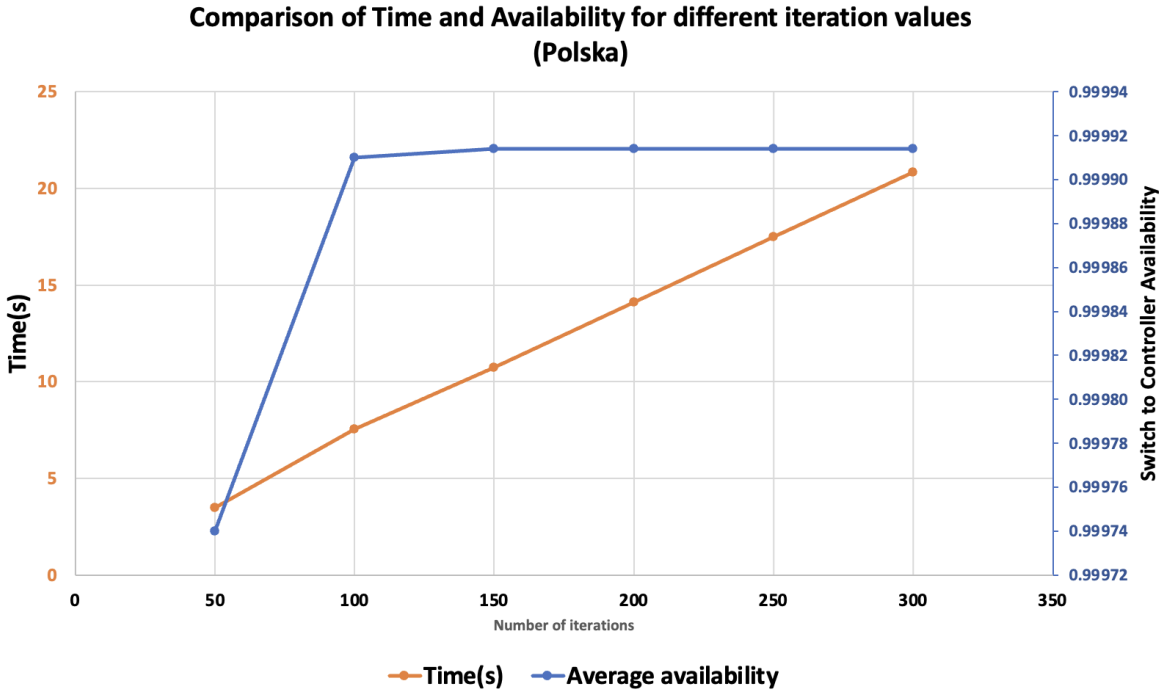


Figure 4.8: Termination = 100 iterations

4.3 Summary

In this chapter, we provided a comprehensive discussion on the Genetic Algorithm (GA) and its methodology. We also highlighted certain significant application areas where the GA has been employed, including optimization problems and machine learning tasks.

Furthermore, we introduced our suggested model for solving the SDN CPP using the GA framework and improving availability. We started by explaining the problem formulation and breakdown of all constraints considered in the model. We then presented a detailed step-by-step description of our GA-based technique, outlining all assumptions, specifications and parameters used in the implementation.

Chapter 5

Experiments and Results

5.1 Dataset

To assess and validate the performance and effectiveness of our heuristic GA-based model, we ran tests and simulations on some networks of the **S**urvivable **N**etwork **D**esign library. The SNDlib [12] is a well established library for survivable fixed telecommunication network design. It is further poised to provide realistic network design test instances to researchers and serve as a standardized baseline for testing, evaluating, and comparing network design models and algorithms.

The dataset currently details data for 22 networks. When combined with a chosen set of planning parameters, this results in a total of 830 unique network planning problem scenarios. Due to this extensive collection of instances, it serves as a go-to for proving the efficacy of novel optimization models and algorithms like ours.

5.2 Test Network Description

The Table 5.1 displays a summary breakdown of the networks we used and their characteristics. These include the number of nodes and edges or links in the network, as well as the average degree of nodes in the particular network, which reflects the mean number of connections that each node has to other nodes. Additionally, the graph diameter, which is the longest shortest path between any two nodes, is also displayed.

Our selection of networks facilitated us in probing the use cases under different scenarios of node densities, enabling us to examine the scalability and robustness of the algorithm in face of augmented complexity. The networks presented steady increase in network nodes and links. Furthermore, the network delay characteristics we used were computed as directly proportional to the corresponding graph diameter. Therefore, carefully selecting networks from the Survivable Network Design library, each demonstrating substantial variances in their respective graph diameters, we crafted a testing environment that enabled us to evaluate under diverse delay constraints. This strategy not only facilitated intra-network analysis but also provided a comparative perspective across different networks.

Polska network displayed in Figure 5.1 falls within the category of networks supplied by actual network service providers and original equipment manufacturers (OEMs). This allowed us run tests that very closely mimicked a real life network.

On the other hand, *Cost*266 and *Germany*50 networks are both among a collection of networks that were established as reference networks within extensive research projects. These projects entailed collaboration between prominent network operators, equipment manufacturers, and academic institutions [56]. Consequently, while the reference networks outlined in the projects may not have depicted actual networks, they

Table 5.1: Topological attributes of the networks

Network	Number of nodes	Number of links	Average node degree	D_g [km]
Polska	12	18	3.00	811
Cost266	37	57	3.08	4032
Germany50	50	88	3.52	4792

were meticulously designed to align with authentic planning situations hence our reason for selecting and working with them. Figure 5.2 and Figure 5.3 provide a visual representation of *Cost*266 and *Germany*50 respectively.

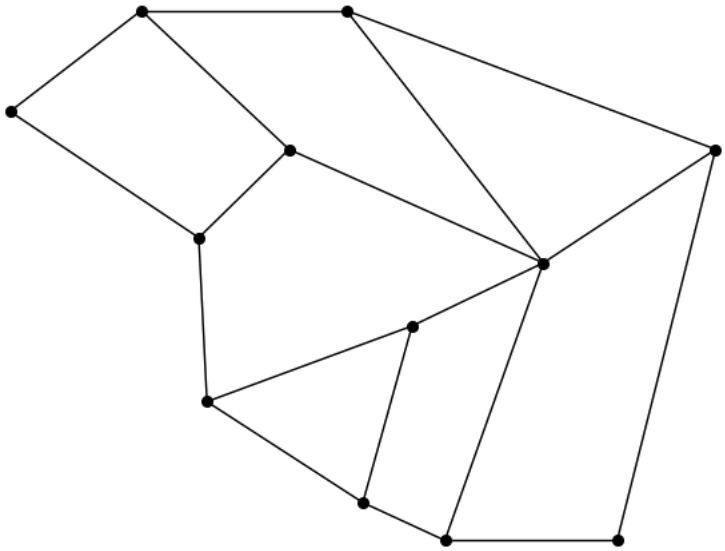


Figure 5.1: Topology of Polska

5.3 Evaluation Measures

We work on the premise that the high threshold delay constraint values D_{sc} and D_{cc} are given as percentages of the graph diameter D_g (longest shortest path between two nodes) per network [51]. This is specified in Table 5.1. Based on this premise, we determined the maximum switch-to-controller (SC) delay value for the polska network to be $D_{\text{sc}} = 35\%$,

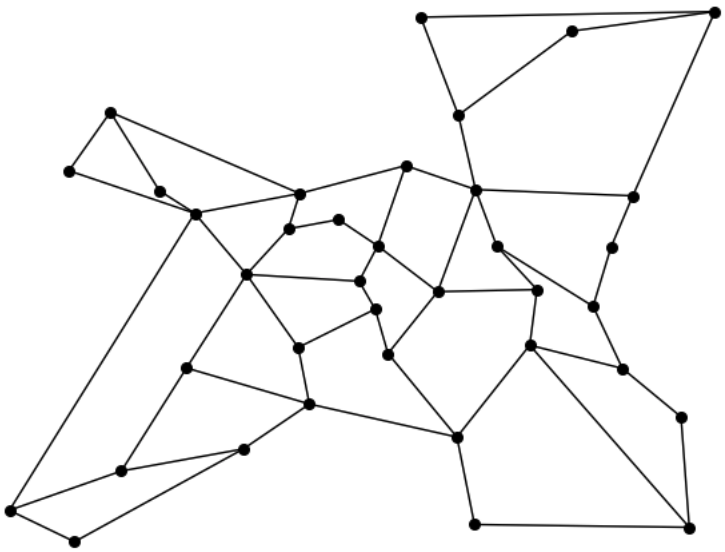


Figure 5.2: Topology of Cost266

for the first iteration and then 40%, 45% and 50%. The maximum controller-to-controller (CC) delay was valued to be $D_{cc} = 70\%$, 75% and then 80% for each round of tests. The same exact parameters were applied to the Cost266 and Germany50 networks.

In our method for finding the optimal number of controllers (C), we begin by assigning C to the smallest value that achieves the given delay criteria. We then progressively increase C in order to preserve our network availability costs as low as feasible. However, the potential drawbacks of having a big number of controllers must be considered. A substantial increase in controllers can result in increased inter-controller communication overhead, which can negatively impact control plane performance and defeat the whole purpose of our CPP. Our model produced very good results and on some rare occasions, we had to incur the cost of adding controller-to-controller restore paths or increasing C to meet our availability threshold.

We demonstrate the trade-off between the number of controllers and the associated availability expenses to facilitate network operators reaching informed decisions regard-

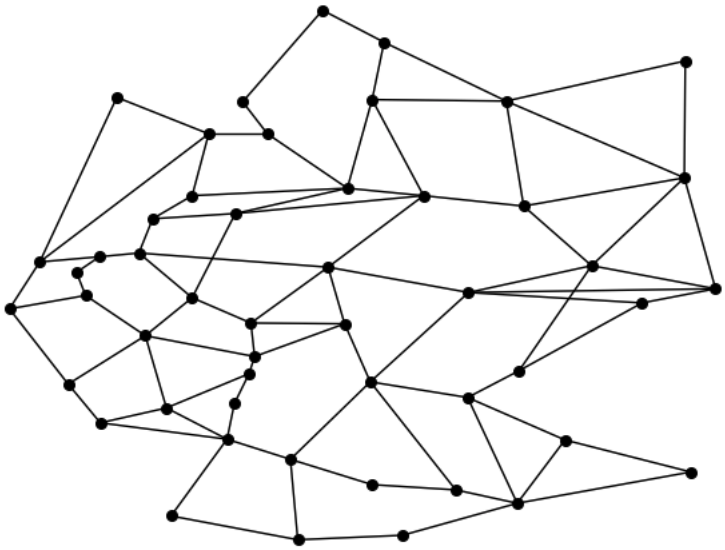


Figure 5.3: Topology of Germany50

ing weighing the benefits and drawbacks of each possible solution. Thus, keeping optimal control plane operation requirements while striking an appropriate balance between controller count and network availability performance.

We have set specific required **effective average availability** values for both inter-controller communication and communication betwwen the data plane and control plane. The target was an effective average availability value of $\lambda_{cc} = \mathbf{0.99999}$ and $\lambda_{sc} = \mathbf{0.9999}$ for the entire network while still respecting the delay constraints. By adhering to these requirements, we aim to maintain a high level of network reliability and resilience, even in the face of potential link or node failures.

5.4 Results, Analysis and Comparison

5.4.1 Evaluation Results

We use python and networkx module [57] to implement our working model. Simulations were run using both jupyter notebook and google colab platforms. Tables 5.2, 5.3 and 5.4 show evaluation results.

Each table showcases the outcomes obtained for various combinations of D_{sc} and D_{cc} values. The 'C' column displays the number of controllers, commencing with the lowest permissible value that attempts to satisfy all targeted constraints, and in some cases, end with a more optimal, minimal cost higher value.

The '*RestorePaths*' column simply indicates the number of costly supplementary restore C-C paths needed to achieve the set threshold availability requirement. Lastly, the S-C and C-C columns are further subdivided into three distinct columns each describing the minimum, maximum and average computed availability for both communications. In the case where a dash or '-' symbol is displayed in these columns, it indicates that the instance is either infeasible – meaning that an appropriate set of controllers cannot be determined for the specified maximum delay values – or that the heuristic was unable to identify a valid solution meeting the constraints. There may be a solution nonetheless.

5.4.2 Analysis

5.4.2.1 Polska

In the case of the Polska network, our model achieved the minimum number of controllers for every instance while satisfying all network delay constraints and attaining the inter-controller availability of five nines without incurring any costs in terms of additional

Table 5.2: Results for Polska network

$D_{\mathbf{sc}}$	$D_{\mathbf{cc}}$	C	*RestorePaths*	**S-C**			**C-C**		
				Min A	**Max A**	**Av A**	**Min A**	**Max A**	**Av A**
35%	70%	-	-	-	-	-	-	-	-
	75%	4	0	0.99980	0.99990	**0.9999**0	0.999997	0.999999	**0.999999**9
	80%	4	0	0.99980	0.99990	**0.9999**0	0.999999	0.999999	**0.999999**9
40%	70%	-	-	-	-	-	-	-	-
	75%	4	0	0.99981	0.99990	**0.9999**0	0.999997	0.999999	**0.999999**9
	80%	4	0	0.99981	0.99990	**0.9999**0	0.999998	0.999999	**0.999999**9
45%	70%	-	-	-	-	-	-	-	-
	75%	4	0	0.99979	0.99998	**0.9999**8	0.999999	0.999999	**0.999999**9
	80%	4	0	0.99979	0.99998	**0.9999**8	0.999999	0.999999	**0.999999**9
50%	70%	2	0	0.99954	0.99994	**0.9999**1	0.999999	0.999999	**0.999999**9
	75%	**2**	0	0.99936	0.99991	**0.9997**8	0.999998	0.999998	**0.999998**8
		3	0	0.99959	0.99999	**0.9999**0	0.999999	0.999999	**0.999999**9
	80%	**2**	0	0.99948	0.99998	**0.9997**9	0.999998	0.999999	**0.999998**8
		3	0	0.99981	0.99999	**0.9999**1	0.999999	0.999999	**0.999999**9

Table 5.3: Results for Cost266 network

$D_{\mathbf{sc}}$	$D_{\mathbf{cc}}$	C	*RestorePaths*	**S-C**			**C-C**		
				Min A	**Max A**	**Av A**	**Min A**	**Max A**	**Av A**
35%	70%	4	2	0.99979	0.99999	**0.99995**	0.999999	0.999999	**0.999995**
	75%	4	2	0.99979	0.99999	**0.99995**	0.999998	0.999999	**0.999995**
	80%	4	2	0.99979	0.99999	**0.99995**	0.999998	0.999999	**0.999995**
40%	70%	3	1	0.99979	0.99996	**0.9999**1	0.999996	0.999999	**0.999999**1
	75%	3	1	0.99979	0.99996	**0.9999**1	0.999994	0.999999	**0.999999**1
	80%	3	1	0.99979	0.99996	**0.9999**1	0.999994	0.999999	**0.999990**
45%	70%	3	0	0.99966	0.99999	**0.99992**	0.999998	0.999999	**0.999998**
	75%	3	0	0.99966	0.99999	**0.99992**	0.999998	0.999999	**0.999998**
	80%	3	0	0.99966	0.99999	**0.99992**	0.999998	0.999999	**0.999998**
50%	70%	2	0	0.99951	0.99995	**0.9997**8	0.999995	0.999999	**0.999996**
		3	0	0.99981	0.99999	**0.9999**1	0.999998	0.999999	**0.999999**9
	75%	2	0	0.99951	0.99995	**0.9997**8	0.999995	0.999999	**0.999996**
		3	0	0.99981	0.99999	**0.9999**1	0.999998	0.999999	**0.999999**9
	80%	2	0	0.99951	0.99995	**0.9997**8	0.999995	0.999999	**0.999996**
		3	0	0.99981	0.99999	**0.9999**1	0.999998	0.999999	**0.999999**9

Table 5.4: Results for Germany50 network

D_{sc}	D_{cc}	C	*RestorePaths*	S-C			C-C		
				Min A	**Max A**	**Av A**	**Min A**	**Max A**	**Av A**
35%	70%	4	2	0.99991	0.99999	**0.99997**7	0.999993	0.999999	**0.999996**
	75%	4	2	0.99991	0.99999	**0.99997**7	0.999993	0.999999	**0.999996**
	80%	3	2	0.99989	0.99999	**0.99996**	0.999993	0.999999	**0.999996**
40%	70%	3	1	0.99991	0.99999	**0.99996**	0.999994	0.999997	**0.999995**
	75%	3	0	0.99991	0.99999	**0.99996**	0.999994	0.999997	**0.999995**
	80%	3	0	0.99991	0.99999	**0.99996**	0.999994	0.999997	**0.999995**
45%	70%	2	0	0.99949	0.99995	**0.99978**	0.999993	0.999993	**0.999997**
		3	0	0.99988	0.99999	**0.99994**	0.999999	0.999999	**0.999999**
	75%	2	0	0.99949	0.99995	**0.99978**	0.999993	0.999993	**0.999994**
		3	0	0.99988	0.99999	**0.99994**	0.999999	0.999999	**0.999999**
	80%	2	0	0.99949	0.99995	**0.99978**	0.999993	0.999993	**0.999996**
		3	0	0.99988	0.99999	**0.99994**	0.999999	0.999999	**0.999999**
50%	70%	2	0	0.99959	0.99999	**0.99988**	0.999998	0.999999	**0.999999**
		3	0	0.99981	0.99999	**0.99991**	0.999999	0.999999	**0.999999**
	75%	2	0	0.99959	0.99999	**0.99988**	0.999998	0.999998	**0.999998**
		3	0	0.99981	0.99999	**0.99991**	0.999999	0.999999	**0.999999**
	80%	2	0	0.99959	0.99999	**0.99988**	0.999998	0.999998	**0.999998**
		3	0	0.99981	0.99999	**0.99991**	0.999999	0.999999	**0.999999**

restore paths. The controller-to-controller restore paths for every instance is zero. However, we had to marginally increase the number of controllers for certain test instances in order to satisfy the four nines constraint for data plane to control plane communications. For example, when D_{sc} is set at 50% and D_{cc} at 75% and 80%, it is noticeable that the initial number of controllers that satisfy all constraints, **except** the four nines for switch-to-controller communication, is 2. By raising the number of controllers to 3, the network is able to meet the requirement for switch-controller communication, while also further enhancing controller-controller availability to six nines and maintaining compliance with delay requirements. This demonstrates the trade-off between the number of controllers and the associated availability expenses to ensure network resilience and also, the effectiveness of the proposed approach in facilitating the network design problem. Figure 5.4 shows the test instance where $D_{sc} = 45\%$ and $D_{cc} = 80\%$. Our model derives

the ideal number of controllers to be 4. This fulfills all delay limits while also providing five nines availability within the control plane and four nines availability between the data and control planes.

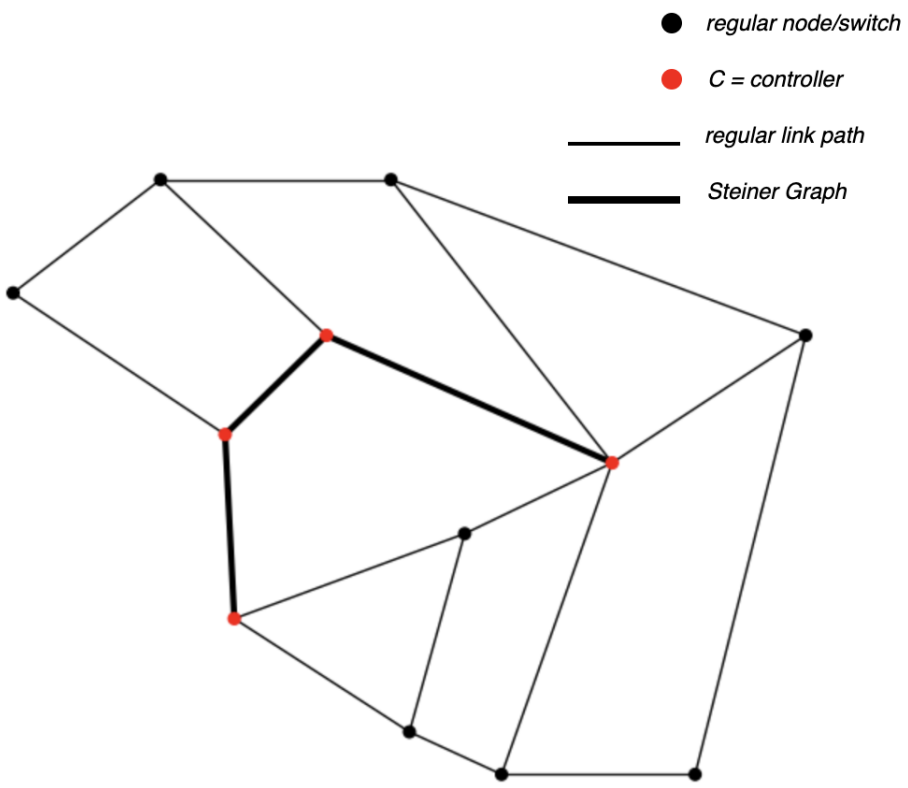


Figure 5.4: Polska Network with Steiner Graph shown in solid lines for C=4, $D_{sc} = 45\%$, $D_{cc} = 80\%$ and $Restorepaths = 0$

5.4.2.2 Cost266

In the Cost266 network, a somewhat inverse linear proportionality is observed between the switch-to-controller (S-C) delay value and the minimum number of computed controllers. When the D_{sc} value is set to the lowest considered value of 35%, the optimal number of controllers is found to be **four**. However, achieving this configuration requires the generation of two controller-to-controller (C-C) restore paths to ensure five

nines availability for inter-controller connectivity. In similar fashion, one restore path is needed when $D_{sc} = 40\%$. Figure 5.5 shows the specific instance where $D_{sc} = 40\%$, $D_{cc} = 70\%$ and $RestorePaths = 1$.

As the D_{sc} value increases, the number of required controllers gradually decreases, eventually reaching a minimum of two. Concurrently, the number of necessary restore paths also diminishes, indicating a reduction and eventual zeroing in the cost associated with achieving five nines availability in the control plane. Interestingly, for the Cost266 network, regardless of the controller-to-controller delay D_{cc} variation per D_{sc}, the cost of attaining five nines availability decreases as D_{sc} increases, leading to less densely connected control networks with fewer controllers. This observation suggests that for this network, less densely connected architectures with fewer controllers can be more cost-effective in achieving high availability in the control plane. Consequently, this insight can inform researchers and network designers in striking an appropriate balance between performance and cost when addressing controller placement and network design in similar scenarios.

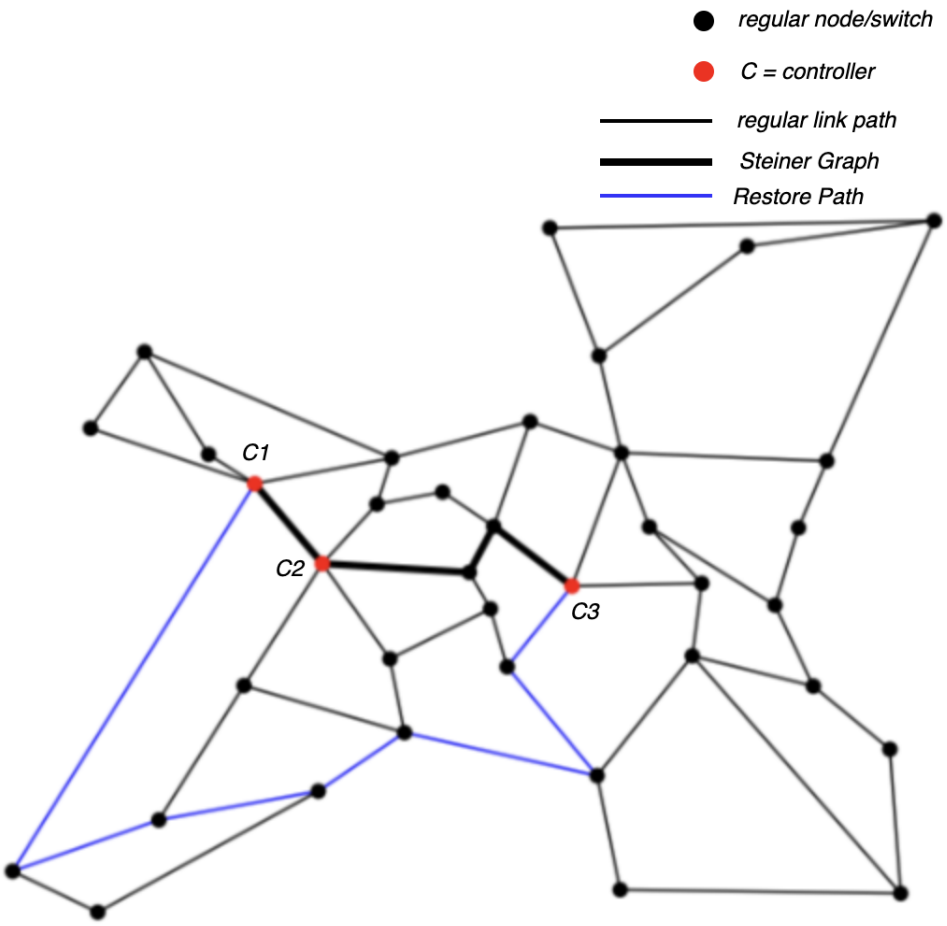


Figure 5.5: Cost266 Network with Steiner Graph shown in solid lines for C=3, $D_{sc} = 40\%$, $D_{cc} = 70\%$ and $Restorepaths = 1$; highlighted in **BLUE** from $C1$ to $C3$

A similar trend can be observed in Cost266 compared to Polska, where for all instances of D_{cc} when D_{sc} is specifically 50%, a marginal increase in the number of controllers by one was sufficient to meet the four nines availability criterion established for data-to-controller plane connections. This observation highlights the importance of balancing the number of controllers with the desired availability level, enabling network designers and researchers to make informed decisions when addressing controller placement and network design in comparable scenarios.

5.4.2.3 Germany50

Finally, we conducted tests on the 50-node network Germany50 as shown in Figure 5.3. When conducting tests with a minimum S-C delay of 35%, we observed once again the largest optimal number of controllers for the network. It came at a cost of restoring 2 paths to align with all stipulations of our network constraints. Figure 5.6 shows a visual of this. With $D_{sc} = 40\%$, only the test case with corresponding $D_{cc} = 70\%$ incurred a cost of restoring 1 path, otherwise requirements were met at zero cost. The Germany50 network emphasizes the idea that an excessive number of controllers in any network can undermine the CPP goals in SDN. A large number of controllers can result in a densely connected control plane, imposing restore path expenses to achieve the required five nines availability.

Slight increment in the number of controllers at $D_{sc} = 45\%$ and 50% was also necessary to hit our four nines mark for S-C connections as every other constraint was met with the initial minimum computed.

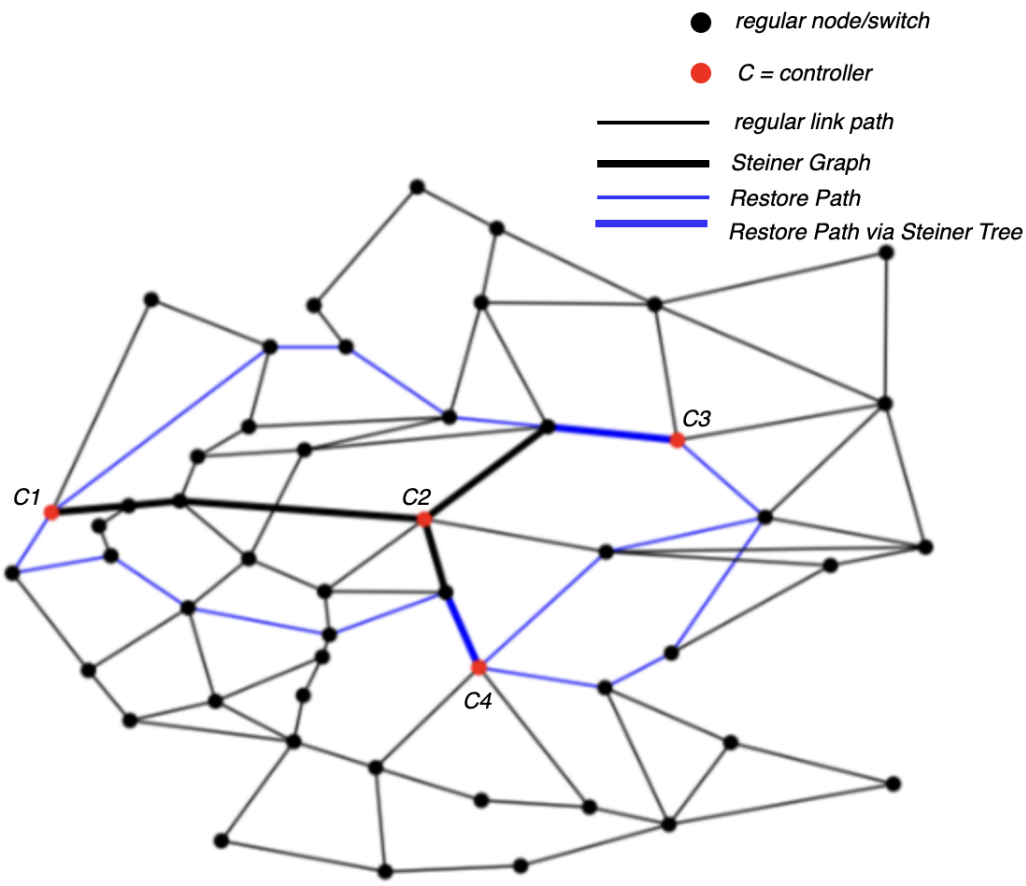


Figure 5.6: Germany50 Network with Steiner Graph shown in solid lines for C=4, $D_{sc} = 35\%$, $D_{cc} = 75\%$ and $Restorepaths = 2$; shown in **BLUE**

5.4.3 Comparison

In this section we compare our work and results to those of comparable studies in order to ascertain the perks of our approach. Santos et al. in both [58] and [52]; and the work by [4] are such.

The approach in [58] addresses the controller placement problem as a bi-objective joint optimization problem encompassing geodiversity and intercontroller availability guarantees. The availability assurances indicate a subgraph with links that can be updated at an expense while the geodiversity restrictions increase resilience to disaster-related failures. They used a heuristic technique to solve what they describes as an NP-complete non-linear issue and tested on Polska and Cost266 networks similar to us. The same team in [52] considered the problem of controller placement under delay boundaries while assuming an increased availability sub-graph between the controllers by upgrades at given costs. They implemented another heuristic and run simulations on similar networks.

5.4.3.1 Runtime

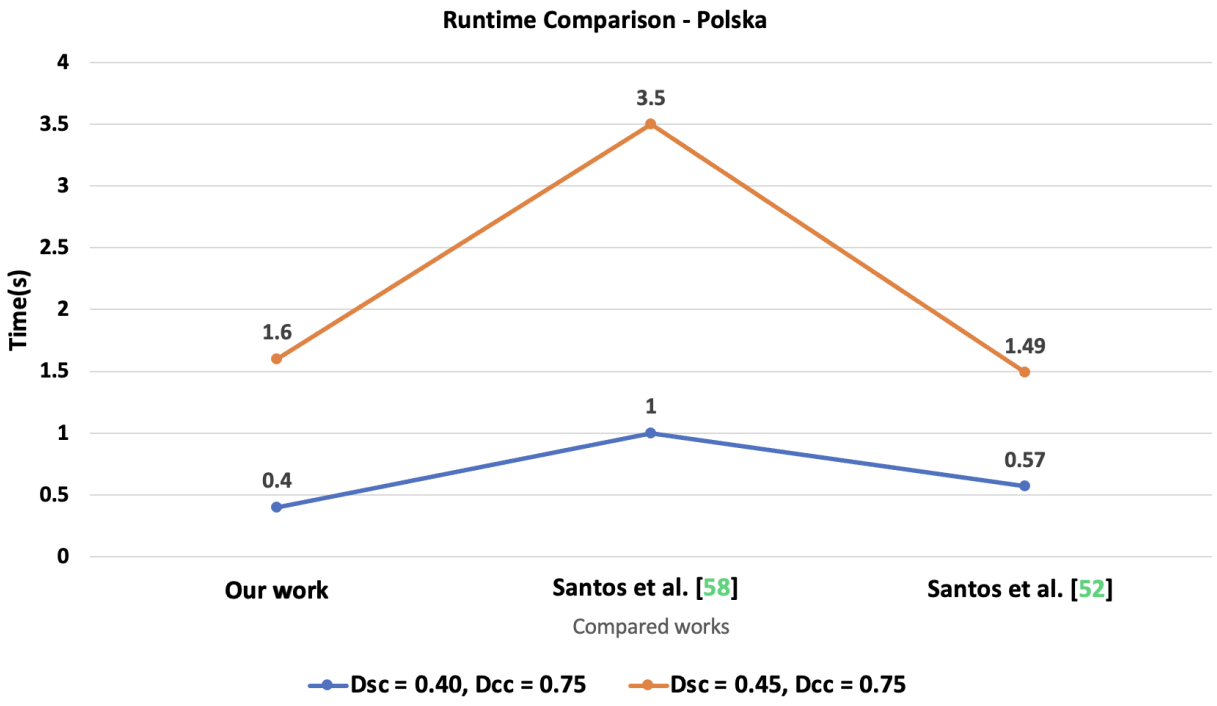


Figure 5.7: Runtime Comparison - Polska

In Figure 5.7, we present a comparison with the work outlined in [58], wherein the same control plane availabilities were examined and identical test instances (specifically D_{sc} at 40% and D_{cc} at 75%, as well as D_{sc} at 45% and D_{cc} at 75%) for the Polska network were employed. Relative to the study mentioned, our approach leveraging Genetic Algorithm (GA) demonstrated a notable increase in computational efficiency, yielding result times of 0.4s and 1.6s as opposed to the documented 1s and 3.5s in [58]. This highlights the accelerated performance of our GA-based methodology.

When compared to the research reported in [52], there is a greater level of competition because the limits under consideration for both techniques are substantially identical. Despite the fact that the approach in [52] gave faster results of 1.49s for the test instance

of D_{sc} at 45% and D_{cc} at 75%, our methodology demonstrated higher performance in the specific test instance of D_{sc} at 40% and D_{cc} at 75% beating their 0.57s. This demonstrates the effectiveness of our technique under a variety of operational restrictions.

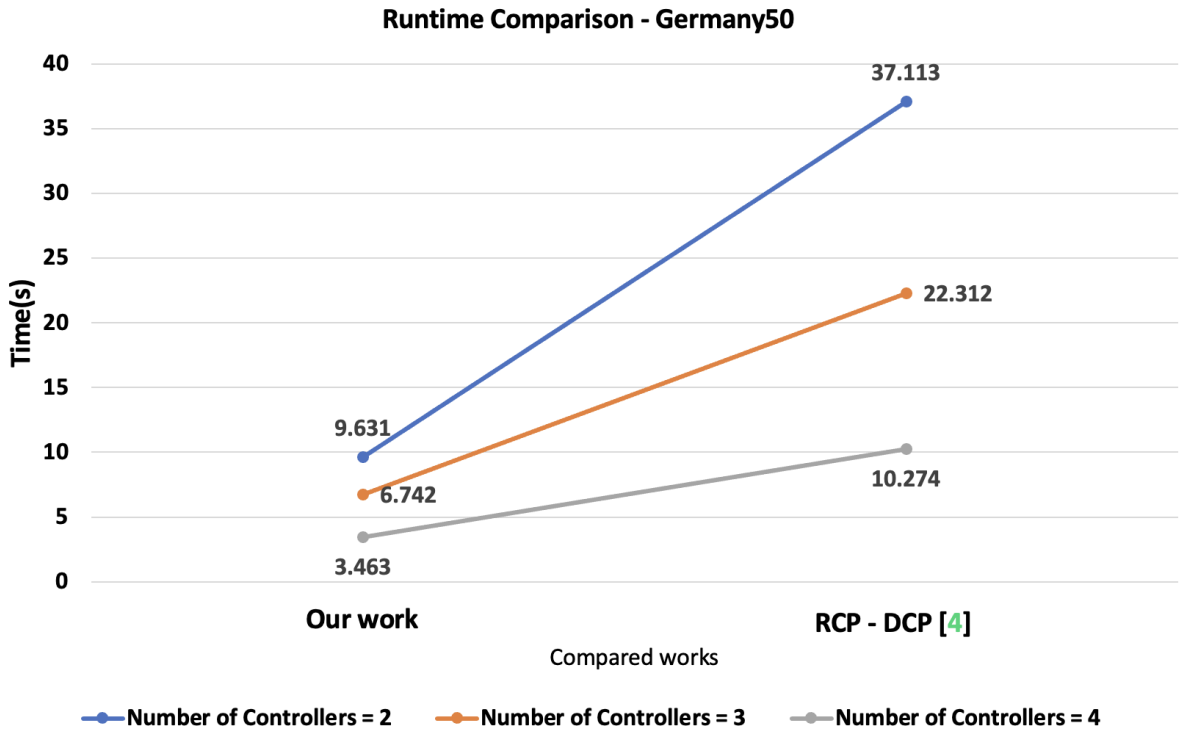


Figure 5.8: Runtime comparison - Germany50

Our experiments conducted on the larger Germany50 network, comprising 50 nodes, can be juxtaposed with the work delineated in [4]. This particular research proposes strategies which are expounded in Section 3.1.3. We selected this paper to serve as our comparative benchmark owing to the shared emphasis on control path availability and reliability. Our comparison is with the RCP-DCP model. Notwithstanding our consideration of additional constraints such as overall network delay, our execution time remains highly competitive. For test instances where D_{sc} is set at 45% and 50%, our algorithm determines an initial controller count of two(2), supplemented by an excep-

tionally efficient average runtime of 9.631 seconds. This compares against the runtime of 37.113 seconds required by the RCP-DCP model. We further achieve 6.742 seconds against 22.312 seconds with a controller count of three(3) to facilitate S-C availability requirements.

Also, the controller count of four(4), ascertained for D_{sc} set at 35%, is achieved post 3.463 seconds, as contrasted to 10.274 seconds demanded by RCP-DCP. RCP-DCP necessitates longer periods because even though the model largely matches the other model in [4], there is an additional constraint mandating the identities of the primary and backup controllers are one and the same.

5.4.3.2 Availability vs Number of controllers

Our analysis has consistently highlighted the importance of balancing the number of controllers with the desired availability level. We compare the results obtained from our approach to that in [58] and the two approaches RCP-DCP and RCP-DCR in [4].

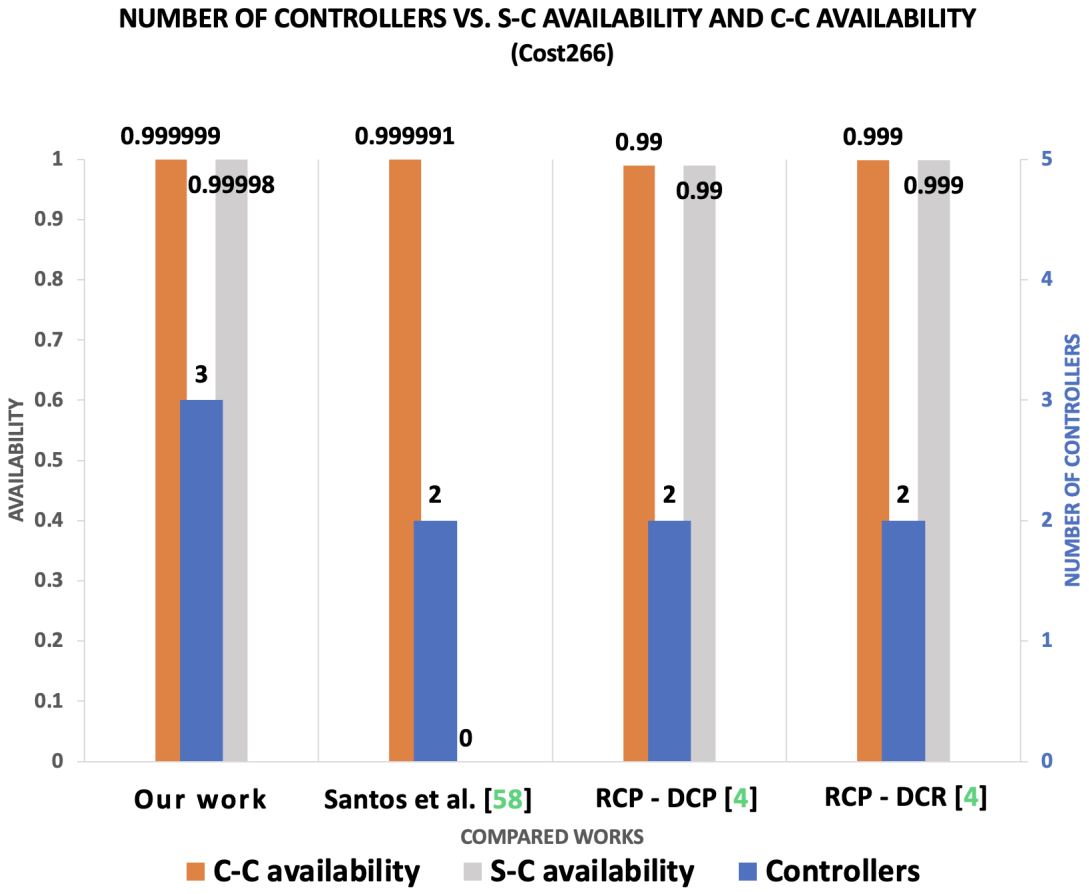


Figure 5.9: Availability comparison

In the equivalent scenario of Cost266, where $D_{sc} = 40\%$ and $D_{cc} = 70\%$, all compared papers converge on the optimum number of controllers being two. In [58], the authors attain a controller-to-controller (C-C) availability of five nines, without giving consideration to the switch-to-controller (S-C) availability. RCP-DCP and RCP-DCR strategies also only manage to reach two and three nines, respectively, for both C-C and S-C availability. Our method introduces the imperative of a trade-off, augmenting the controller count to three, enabling the achievement of five nines in C-C and four nines in S-C availability, hence promoting long-term overall availability and cost efficiency.

Chapter 6

Conclusion and Future Work

This concluding chapter summarizes our research work done and identifies future projects based on our contributions that have yet to be sought out. Section 6.1 briefly discusses the deductions and conclusions drawn from Chapters 4 and 5. Section 6.2 outlines our suggestions for future work directions.

6.1 Summary

This work focuses on optimizing the control plane design by determining the controller placement for minimum delay and substantially enhancing the switch to controller as well as controller to controller link availability. We present an Enhanced Genetic Algorithm approach and demonstrate an exact method for optimizing the inherent availability constraints. We conducted computational tests on three networks of varied sizes to assess the trade-off between the number of controllers deployed and the cost of increasing specific node link availabilities.

From the network operator's perspective, minimizing the number of controllers is desirable to reduce inter-controller communication overhead. However, this typically

results in larger switch to controller delays and decreased control path availability, necessitating more link availability improvements, thereby increasing costs. Deploying a few additional controllers can significantly reduce cost, stabilize the network and maintain the node to node communication without seriously compromising switch to controller or inter-controller communication overhead. However, having too many controllers can not only compromise communication overhead but also lead to increased availability costs, as presented in the computational results.

6.2 Future Work

Our heuristic approach has yielded promising outcomes, confirming the necessity for striking a balance between the quantity of controllers, delay constraints, and path availability. We propose the following recommendations for future research as worthwhile avenues to explore:

- Testing on larger and more elastic networks

  Evaluating our model on larger and more elastic networks, would enable us to better understand the scalability and robustness of our approach. By exploring networks with a diverse range of topologies and sizes, we can gain more knowledge about how our model performs under various real-world conditions and refine it accordingly.

- Testing with additional constraints

  Incorporating supplementary requirements such as load balancing and security, may be worth looking into, particularly when evaluating our model on larger networks.

References

[1] Paul Goransson, Chuck Black, and Timothy Culver. *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[2] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[3] Tamal Das and Mohan Gurusamy. Resilient controller placement in hybrid sdn/legacy networks. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2018.

[4] Petra Vizarreta, Carmen Mas Machuca, and Wolfgang Kellerer. Controller placement strategies for a resilient sdn control plane. In *2016 8th international workshop on resilient networks design and modeling (RNDM)*, pages 253–259. IEEE, 2016.

[5] T. A. Al-Janabi and H. S. Al-Raweshidy. Efficient whale optimisation algorithm-based SDN clustering for IoT focused on node density. In *16th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, 2017.

[6] Ahmed Abdelaziz, Ang Tan Fong, Abdullah Gani, Usman Garba, Suleman Khan, Adnan Akhunzada, Hamid Talebian, and Kim-Kwang Raymond Choo. Distributed

controller clustering in software defined networks. *PLOS ONE*, 12(4):e0174715, 4 2017.

[7] Crossover in Genetic Algorithm - GeeksforGeeks.

[8] Tanweer Alam, Shamimul Qamar, Amit Dixit, and Mohamed Benaida. Genetic algorithm: Reviews, implementations, and applications. *CoRR*, abs/2007.12673, 2020.

[9] Rong Yujie, Wu Muqing, and Chen Yiming. An Effective Controller Placement Algorithm Based on Clustering in SDN. In *IEEE 6th International Conference on Computer and Communications (ICCC)*, pages 2294–2299, 2020.

[10] Jonathan Shapiro. *Genetic Algorithms in Machine Learning*, pages 146–168. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[11] David Hock, Matthias Hartmann, Steffen Gebert, Michael Jarschel, Thomas Zinner, and Phuoc Tran-Gia. Pareto-optimal resilient controller placement in sdn-based core networks. In *Proceedings of the 25th International Teletraffic Congress (ITC)*, pages 1–9, 2013.

[12] S Orlowski, Michal Pioro, A Tomaszewski, and R Wessäly. Sndlib 1.0 - survivable network design library. *Networks*, 55(3):276–286, 2009.

[13] Albert Greenberg, Gisli Hjalmtysson, David A Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54, 2005.

[14] Tamal Das, Vignesh Sridharan, and Mohan Gurusamy. A survey on controller placement in sdn. *IEEE Communications Surveys Tutorials*, 22(1):472–503, 2020.

[15] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1):236–262, 2015.

[16] Bassey Isong, Reorapetse Ramoliti Samuel Molose, Adnan M Abu-Mahfouz, and Nosipho Dladlu. Comprehensive review of sdn controller placement strategies. *IEEE Access*, 8:170070–170092, 2020.

[17] Hlabishi I Kobo, Adnan M Abu-Mahfouz, and Gerhard P Hancke. A survey on software-defined wireless sensor networks: Challenges and design requirements. *IEEE access*, 5:1872–1899, 2017.

[18] Yassine Maleh, Youssef Qasmaoui, Khalid El Gholami, Yassine Sadqi, and Soufyane Mounir. A comprehensive survey on sdn security: threats, mitigations, and future directions. *Journal of Reliable Intelligent Environments*, pages 1–39, 2022.

[19] Murat Karakus and Arjan Durresi. A survey: Control plane scalability issues and approaches in software-defined networking (sdn). *Computer Networks*, 112:279–293, 2017.

[20] Abdelhamied A Ateya, Ammar Muthanna, Anastasia Vybornova, Abeer D Algarni, Abdelrahman Abuarqoub, Y Koucheryavy, and Andrey Koucheryavy. Chaotic salp swarm algorithm for sdn multi-controller networks. *Engineering Science and Technology, an International Journal*, 22(4):1001–1012, 2019.

[21] Vahid Ahmadi and Mostafa Khorramizadeh. An adaptive heuristic for multi-objective controller placement in software-defined networks. *Computers & Electrical Engineering*, 66:204–228, 2018.

[22] Yongli Zhao, Chuan Liu, Hua Wang, Xinfeng Fu, Qi Shao, and Jie Zhang. Load balancing-based multi-controller coordinated deployment strategy in software defined optical networks. *Optical Fiber Technology*, 46:198–204, 2018.

[23] Mostafa Khorramizadeh and Vahid Ahmadi. Capacity and load-aware software-defined network controller placement in heterogeneous environments. *Computer Communications*, 129:226–247, 2018.

[24] Jie Lu, Zhen Zhang, Tao Hu, Peng Yi, and Julong Lan. A survey of controller placement problem in software-defined networking. *IEEE Access*, 7:24290–24307, 2019.

[25] Jianxin Liao, Haifeng Sun, Jingyu Wang, Qi Qi, Kai Li, and Tonghong Li. Density cluster based approach for controller placement problem in large-scale software defined networkings. *Computer Networks*, 112:24–35, 2017.

[26] Shuai Wu, Xiaoqian Chen, Lei Yang, Chengguang Fan, and Yong Zhao. Dynamic and static controller placement in software-defined satellite networking. *Acta Astronautica*, 152:49–58, 2018.

[27] Afrim Sallahi and Marc St-Hilaire. Optimal model for the controller placement problem in software defined networks. *IEEE communications letters*, 19(1):30–33, 2014.

[28] Afrim Sallahi and Marc St-Hilaire. Expansion model for the controller placement problem in software defined networks. *IEEE Communications Letters*, 21(2):274–277, 2016.

[29] Jianxin Liao, Haifeng Sun, Jingyu Wang, Qi Qi, Kai Li, and Tonghong Li. Den-

sity cluster based approach for controller placement problem in large-scale software defined networkings. *Computer Networks*, 112:24–35, 1 2017.

[30] Amin Shahraki, Amir Taherkordi, Øystein Haugen, and Frank Eliassen. A survey and future directions on clustering: From wsns to iot and modern networking paradigms. *IEEE Transactions on Network and Service Management*, 18(2):2242–2274, 2021.

[31] Hadar Sufiev, Yoram Haddad, Leonid Barenboim, and José Soler. Dynamic SDN controller load balancing. *Future Internet*, 11(3), 2019.

[32] Moushita Patnaik and Angelia Melani Adrian. A perspective depiction of heuristics in virtual reality. In *Cognitive Big Data Intelligence with a Metaheuristic Approach*, pages 101–116. 2022.

[33] Abeer A.Z. Ibrahim, Fazirulhisyam Hashim, Nor K. Noordin, Aduwati Sali, Keivan Navaie, and Saber M.E. Fadul. Heuristic Resource Allocation Algorithm for Controller Placement in Multi-Control 5G Based on SDN/NFV Architecture. *IEEE Access*, 9:2602–2617, 2021.

[34] Kshira Sagar Sahoo, Anamay Sarkar, Sambit Kumar Mishra, Bibhudatta Sahoo, Deepak Puthal, Mohammad S. Obaidat, and Balqies Sadun. Metaheuristic solutions for solving controller placement problem in SDN-based WAN architecture. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications*, pages 15–23, 2017.

[35] A Vybornova. A Survey on the Swarm Intelligence Approaches to Controller Placement Problem in the Software Defined Networks Design and Optimization. *Telecom IT*, 8(4):83–92, 12 2020.

[36] Stanislav Lange, Steffen Gebert, Thomas Zinner, Phuoc Tran-Gia, David Hock, Michael Jarschel, and Marco Hoffmann. Heuristic approaches to the controller placement problem in large scale SDN networks. *IEEE Transactions on Network and Service Management*, 12(1):4–17, 3 2015.

[37] Piotr Czyżżak and Adrezej Jaszkiewicz. Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of multi-criteria decision analysis*, 7(1):34–47, 1998.

[38] Ashutosh Kumar Singh and Shashank Srivastava. A survey and classification of controller placement problem in sdn. *International Journal of Network Management*, 28(3):e2018, 2018.

[39] John H. Holland. The Robustness of Genetic Plans. In *Adaptation in Natural and Artificial Systems*. The MIT Press, 1992.

[40] Oliver Kramer. In *Genetic Algorithm Essentials*. Springer, 2017.

[41] Mitchell Melanie. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[42] Noraini Mohd Razali, John Geraghty, et al. Genetic algorithm performance with different selection strategies in solving tsp. In *Proceedings of the world congress on engineering*, volume 2, pages 1–6. International Association of Engineers Hong Kong, China, 2011.

[43] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *CoRR*, abs/1109.3627, 2011.

[44] I. Abuiziah and N. Shakarneh. A review of genetic algorithm optimization: Operations and applications to water pipeline systems. *International Journal of Mathematical and Computational Sciences*, 7(12):1782 – 1788, 2013.

[45] Tanweer Alam. Design a blockchain-based middleware layer in the Internet of Things Architecture. *JOIV : International Journal on Informatics Visualization*, 4(1), 2 2020.

[46] Arthur C. Brooks. Genetic Algorithms and Public Economics. *Journal of Public Economic Theory*, 2(4):493–513, 10 2000.

[47] Rahul Kala, Anupam Shukla, and Ritu Tiwari. Robotic path planning using evolutionary momentum-based exploration. *Journal of Experimental & Theoretical Artificial Intelligence*, 23(4):469–495, 12 2011.

[48] Rita Girão-Silva, Lúcia Martins, Teresa Gomes, Abdulaziz Alashaikh, and David Tipper. Improving network availability—a design perspective. In Xin-She Yang, Simon Sherratt, Nilanjan Dey, and Amit Joshi, editors, *3rd International Congress on Information and Communication Technology*, pages 799–815, 2019.

[49] Brandon Heller, Rob Sherwood, and Nick Mckeown. The controller placement problem. *Computer Communication Review*, 42(4):473–478, 2012.

[50] Dorabella Santos, Teresa Gomes, and David Tipper. Software-Defined Network Design driven by Availability Requirements. In *16th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2020.

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

[52] Dorabella Santos, Joao P. Vidal, Teresa Gomes, and Lucia Martins. A Heuristic Method for Controller Placement and Enhanced Availability between SDN Con-

trollers. In *Proceedings of the 11th International Conference on Network of the Future (NoF)*, pages 82–90, 2020.

[53] P. Cholda. Network Recovery, Protection and Restoration of Optical, SONET-SDH, IP, and MPLS. *IEEE Communications Magazine*, 43(7):12–12, 7 2005.

[54] Francisco Javier Ros and Pedro Miguel Ruiz. Five nines of southbound reliability in software-defined networks. In *Proceedings of the 3rd workshop on hot topics in software defined networking*, pages 31–36, 2014.

[55] Blake Thorne. Four nines and beyond: A guide to high availability infrastructure, 2018.

[56] M. Gunkel, R. Leppla, M. Wade, A. Lord, D. Schupke, G. Lehmann, C. Fürst, S. Bodamer, B. Bollenz, H. Haunstein, H. Nakajima, and J. Martensson. A cost model for the WDM layer. In *2006 International Conference on Photonics in Switching, Proceedings, PS*, pages 108–113, 2006.

[57] NetworkX — NetworkX documentation.

[58] Dorabella Santos, Teresa Gomes, Lúcia Martins, and João P Vidal. Resilient sdn intercontroller network design under availability requirements and geodiversity constraints. In *2022 18th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 1–8. IEEE, 2022.