Resilient Controller Placement Problems in Software Defined Wide-Area Networks

by

Maryam Tanha
B.Sc., Yazd University, Iran, 2005
M.Sc., Universiti Putra Malaysia, 2014

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Resilient Controller Placement Problems in Software Defined Wide-Area Networks

by

Maryam Tanha
B.Sc., Yazd University, Iran, 2005
M.Sc., Universiti Putra Malaysia, 2014

Supervisory Committee

---

Dr. Jianping Pan, Supervisor
(Department of Computer Science)

---

Dr. Sue Whitesides, Departmental Member
(Department of Computer Science)

---

Dr. Issa Traore, Outside Member
(Department of Electrical and Computer Engineering)

## ABSTRACT

Software Defined Networking (SDN) is an emerging paradigm for network design and management. By providing network programmability and the separation of control and data planes, SDN offers salient features such as simplified and centralized management, reduced complexity, and accelerated innovation. Using SDN, the control and management of network devices are performed by centralized software, called controllers. In particular, Software-Defined Wide Area Networks (SD-WANs) have made considerable headway in recent years. However, SDN can be a double-edged sword with regard to network resilience. The great reliance of SDN on the logically centralized control plane has heightened the concerns of research communities and industries about the resilience of the control plane. Although the controller provides flexible and fine-grained resilience management features that contribute to faster and more efficient failure detection and containment in the network, it is the Achilles' heel of SDN resilience. The resilience of control plane has a great impact on the functioning of the whole system. The challenges associated with the resilience of control plane should be addressed properly to benefit from SDN's unprecedented capabilities.

This dissertation investigates the aforementioned issues by categorizing them into two groups. First, the resilient design of the control plane is studied. The resilience of the control plane is strongly linked to the *Controller Placement Problem* (CPP), which deals with the positioning and assignment of controllers to the forwarding devices. A resilient CPP needs to assign more than one controller to a switch while it satisfies certain Quality of Service (QoS) requirements. We propose a solution for such a problem that, unlike most of the former studies, takes both the switch-controller/inter-controller latency requirements and the capacity of the controllers into account to meet the traffic loads of switches. The proposed algorithms, one of which has a polynomial-time complexity, adopt a *clique-based* approach in graph theory to find high-quality solutions heuristically.

Second, due to the high dynamics of SD-WANs in terms of variations in traffic loads of switches and the QoS requirements that further affect the incurred load on the controllers, adjustments to the controller placement are inevitable over time. Therefore, *resilient switch reassignment* and *incremental controller placement* are proposed to reuse the existing geographically distributed controllers as much as possible or make slight modifications to the controller placement. This assists the service providers in decreasing their operational and maintenance costs. We model these

problems as variants of the problem of *scheduling on parallel machines* while considering the capacity of controllers, reassignment cost, and resiliency (which have not been addressed in the existing research work) and propose approximation algorithms to solve them efficiently.

To sum up, CPP has a great impact on the resilience of SDN control plane and subsequently the correct functioning of the whole network. Therefore, tailored mechanisms to enhance the resiliency of the control plane should be applied not only at the design stage of SD-WANs but also during their lifespan to handle the dynamics and new requirements of such networks over time.

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| ACO | Ant Colony Optimization |
| API | Application Programming Interface |
| BFS | Breadth First Search |
| BFT | Byzantine Fault Tolerant |
| BRSRP | Budgeted Resilient Switch Reassignment Problem |
| CNCP | Capacitated Next Controller Placement |
| CPP | Controller Placement Problem |
| CRED | Center for Research on the Epidemiology of Disasters |
| DCPP | Dynamic Controller Provisioning Problem |
| DFS | Depth First Search |
| DoS | Denial of Service |
| GR | GReedy |
| GSD | Greedy Switch Dissociation |
| ICPP | Incremental Controller Placement Problem |
| ICT | Information and Communication Technology |
| IoT | Internet of Things |
| IP | Internet Protocol |
| NFV | Network Function Virtualization |
| POCO | Pareto-Optimal Controller Placement |
| PoP | Point of Presence |
| QoS | Quality of Service |
| OSPF | Open Shortest Path First |
| RCCPP | Resilient Capacitated Controller Placement Problem |
| SDN | Software Defined Networking |
| SD-WAN | Software-Defined Wide Area Network |
| SDRBG | Switch Dissociation and Reassignment with Bipartite Graph rounding |
| SR | Switch Reassignment |

| | |
|---|---|
| SRBG | Switch Reassignment with Bipartite Graph rounding |
| SLA | Service Level Agreement |
| URSRP | Unbudgeted Resilient Switch Reassignment Problem |
| VM | Virtual Machine |

# ACKNOWLEDGEMENTS

There are a number of people to whom I am greatly indebted. I would like to express my deep gratitude to my supervisor, Prof. Jianping Pan, for his generous support and great encouragement to conduct this research as well as his valuable comments to enhance the quality of the dissertation. Also, I am very grateful to the members of my supervisory committee, Prof. Sue Whitesides and Prof. Issa Traore, for their help and support. Finally, I have a sincere gratitude to all of those with whom I have had the pleasure to work during my PhD program as well as my fellow lab mates.

# DEDICATION

This dissertation is dedicated to my dearest husband, my closest friend, my fellow class mate, lab mate and many more, Dawood, for his whole-hearted and substantial support, as well as to my loving parents and siblings.

# Chapter 1

# Introduction

## 1.1 Background

Nowadays, communication networks are indispensable to all sectors of society. Many critical infrastructures such as smart grid, transportation systems, and health systems are greatly dependent on Information and Communication Technologies (ICTs). Thus, the malfunction or failure of communication systems resulting from various challenges would have devastating impacts on the normal operation of societies. Such challenges include accidental mis-configuration or operational mistakes, large-scale natural or human-made disasters (e.g., earthquakes, floods, electromagnetic pulses, and bombs), malicious attacks (e.g., security threats), environmental conditions (e.g., mobility, constrained resources, and volatile traffic loads), and so on. In addition to human losses, natural disasters incur high economic costs. The report published by the Center for Research on the Epidemiology of Disasters (CRED) in 2017 shows an emerging trend in natural disaster events with a higher incurred cost. In particular, economic losses were $334 billion in 2017 compared to $142 billion between 2007 and 2016 [1].

In large-scale and complex backbone networks, multiple correlated and cascaded failures can cause widespread connectivity losses and affect many mission-critical applications and services [2] [3] that public safety agencies rely on. For instance, the great east Japan earthquake in 2011 and its resultant tsunami caused serious damages to a lot of communication infrastructures. Telecom switching offices, optical fiber links, and base stations for mobile services were completely or partially damaged. This resulted in an explosive growth in the demand for ICT services due to the fact

that the inhabitants of the affected areas desperately sought to communicate with the outside world which subsequently led to traffic congestion in the network [4]. All the aforementioned challenges pose dire threats to the resilience of the communication networks and subsequently on their dependent crucial infrastructures. To have a clearer definition for resilience, we refer to the following description: "*Resilience is the ability of the network to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation.*" [3]. In particular, resilience disciplines include survivability, traffic tolerance, disruption tolerance, dependability, security, and performability whose details are provided in [5].

The evolution of the Internet's physical infrastructure, its protocols and performance, has become extremely demanding due to its rapid growth and large-scale deployment [6]. Moreover, emerging technologies such as Internet of Things (IoT), Cloud Computing, and Big Data underscore the need for faster, more scalable, efficient, and resilient network architectures. As a trend towards network softwarization by the separation of control and data planes in a Software Defined Networking (SDN) architecture, the network intermediate nodes (called switches) are simple forwarding devices without having a complicated control plane which results in more cost-efficient devices. Figure 1.1 illustrates the SDN architecture. The application layer includes business applications such as network virtualization and security applications that utilize the network services. The software-based controllers are located in the control layer (i.e., control plane layer) and are in charge of the management and control of network devices using open standards and protocols. The north-bound Application Programming Interfaces (APIs) allow for communication between application layer and control layer to provision typical network services such as routing, traffic engineering, security, access control, and Quality of Service (QoS) [7]. The south-bound open interfaces provide the communication channel between the control plane and data plane layer (also, called the infrastructure layer). The most well-known protocol for the southbound communication interface is OpenFlow [8], and it is considered as a critical building block for almost all of the SDN solutions.

However, SDN is a double-edged sword with regard to network resilience. While it shows promise to enhance the resilience of the communication networks, it underscores the need for a more resilient communication system and introduces new menaces to network resilience. SDN decouples the control plane from the data plane, and the key decision making and management functions are carried out by a (logically) centralized entity called a controller. The controller provides flexible and fine-grained resilience

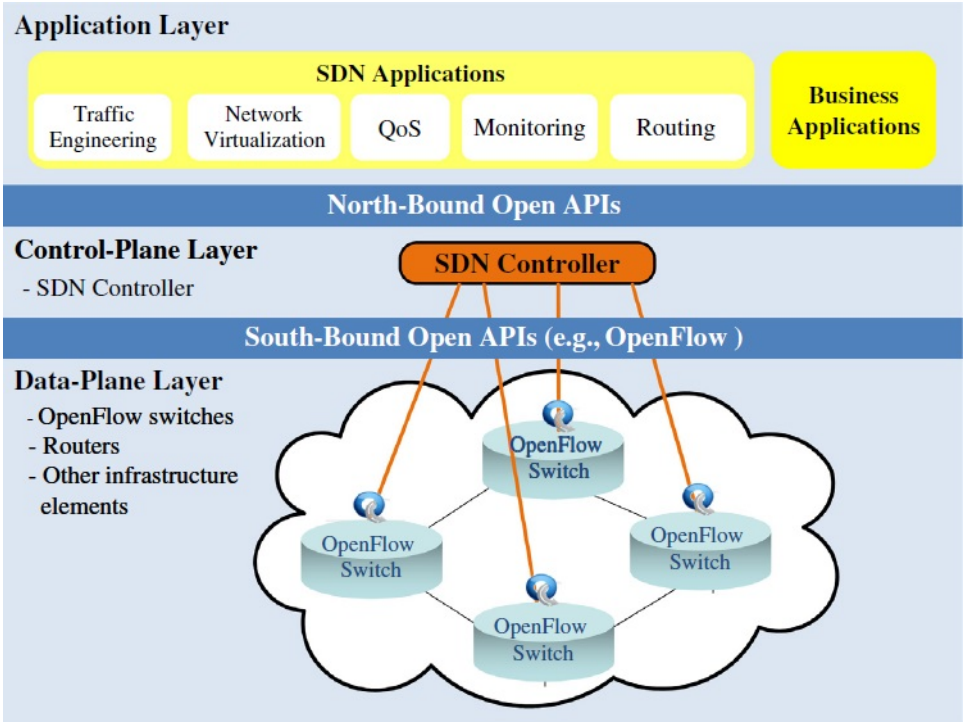



Figure 1.1: SDN architecture [6].

management features that contribute to faster and more efficient failure detection and containment in the network; however, it is the Achilles' heel of SDN resilience since its failure would affect the proper functionality of the whole network. Therefore, the resiliency of the control plane in an SDN-based network is of great significance and it is the main focus of this dissertation.

Furthermore, it is worth mentioning that the full deployment of SDN in existing WANs presents economic, organizational, and technical challenges [9]. Thus, some network operators prefer an incremental migration to SDN over time (i.e., a multi-period planning for the deployment of SDN-enabled devices in existing networks) while they aim to maximize the benefits that SDN brings to their network through enhancing traffic engineering flexibility. Our carried out research with regard to progressive migration to SDN while improving traffic engineering flexibility for handling networks dynamics (such as link/node failures) in the data plane has been included in Appendix A.

## 1.2 Research Objectives

In spite of the great virtues of utilizing SDN solutions, there are some open issues and challenges that should be addressed to benefit from the unprecedented features of SDN. More specifically, we are interested in resilience issues in Software-Defined Wide Area Networks (SD-WANs), which serve as the software-defined backbones that connect Points of Presence (PoPs), as follows:

**Resilient Design of the Control Plane:** Due to the fact that controllers are the heart of SDN functionality, they are the main resilience bottlenecks. Controllers may fail randomly resulting from natural disasters, power outages, security bugs, and malicious/terrorist attacks. These disruptive events may cause adverse impacts on the network infrastructure by (partially) demolishing controller instances in a geographical area and subsequently affecting many network applications and services. Therefore, the SDN control plane requires a high level of resilience that is tightly interwoven with the *Controller Placement Problem* (CPP), which is NP-hard [10]. It indicates the number of required controllers to handle the traffic loads of switches (mainly the number of flow setup requests) as well as their location (in the network topology) in an efficient and cost-effective manner. Resilient controller placement contributes towards mitigating the impact of controller failures by assigning multiple controllers to a switch. Moreover, QoS requirements, including switch-controller and inter-controller latency thresholds must be satisfied. Such latency bounds are of utmost importance since they impact the control decision and the satisfaction of Service Level Agreements (SLAs). For instance, in SD-WANs, the round trip propagation latency between the switch and each of its associated controllers must be less than or equal to 50 ms [11]. It should be noted that latency between a switch and its assigned controller as well as the inter-controller latency consists of different components, including transmission latency, processing and queuing latency at the controller, and propagation latency. However, in SD-WANs, propagation latency is the main contributor to the total latency (due to the large geographical distance among the PoPs) and other types of latencies are negligible. In addition, switch-controller latency and inter-controller latency can be affected by network conditions such as congestion. This is usually addressed by incorporating proper traffic engineering mechanisms such as prioritizing control traffic over data traffic. Another important factor in resilient controller placement is the capacity of controller. Each

controller can only handle a limited number of flow set up requests per second due to its resource constraints. An overloaded controller would have a higher probability of failure as well as causing the processing latency to increase, which subsequently affects the switch-controller latency. Therefore, considering a resilient capacitated controller placement is a better representation of real-life applications in contrast to the uncapacitated version (i.e., assuming unlimited capacity for a controller). Few existing work has considered all the aforementioned factors simultaneously or has provided scalable solutions to resilient CPP. Furthermore, there are trade-offs and interdependencies among the aforementioned factors. Although no single best placement strategy exists, having a flexible design especially in terms of switch-controller latency and inter-controller latency assists in achieving a balanced trade-off among different factors of controller placement. For instance, placing controllers close to each other reduces the controller communication cost (latency) but it causes switch-controller latency to rise. Also, connecting switches to their nearest controller(s) may lead to load imbalance among the controllers, and subsequently increase latency due to queuing time at some of the controllers. Therefore, by using latency bounds (rather than minimizing the total switch-controller and/or inter-controller latencies), we can achieve a better trade-off among different factors. It also results in having a more inclusive design since it can be easily converted to the case where the total switch-controller latency is minimized by reducing the latency threshold as much as possible. In addition, it should be noted that a flat SDN control architecture [12], as a common distributed SDN control architecture, requires a peer-to-peer communication among the controllers [13]. Regardless of the methods utilized to manage the network state consistency, the connectivity among controllers determines the maximum time required to update information among them [14]. Finally, the objective of the static resilient CPP is to minimize the cost in terms of the number of controllers (since we assign multiple controllers to each switch), which subsequently decreases the operational and maintenance costs.

**Resilient Switch Reassignment and Incremental Controller Placement:** While having a resilient control plane through a resilient controller placement at the design stage of the network is essential, maintaining this resiliency over time with regard to different changes in the system is also of utmost importance and it has been less explored in the existing work. Adjustments to the controller placement are inevitable due to the high dynamics of SD-WANs in terms of variations in traffic

loads of switches and the change in QoS requirements that further affect the incurred load on the controllers. The service providers prefer to reuse the existing geographically distributed controllers as much as possible or make slight modifications to the controller placement to decrease their operational and maintenance costs. To achieve this, resilient switch reassignments and incremental controller placement for a given controller placement should be investigated. Resilient switch reassignment should maintain the resiliency of the system while satisfying the QoS requirements. Also, it assists in having a better load balancing among the controllers, which subsequently leads to achieving traffic tolerance as another resilience discipline. However, if switch reassignment is not sufficient, adding a number of controllers to the current set of controllers incrementally helps maintain the resiliency of the system. In particular, switch reassignment may not be able to prevent the violation of controller capacity resulting from the change in traffic loads of switches or to avoid the infeasibility of the current controller placement due to the change in switch-controller latency threshold. In both aforementioned cases, it is required to amend the set of current controllers with a number of new controllers to meet the new conditions. Another important factor in switch reassignment is the instability of the system which may result from the high frequency of reassignments. The frequency of switch reassignment can be controlled by considering appropriate time intervals to perform the reassignment. Furthermore, switch reassignment can cause service disruption. However, possible disruptions can be handled by having a disruption-free switch reassignment protocol such as [15] in place. More details are given in Chapter 3.

## 1.3 Contributions

Following the identified research objectives with regard to the resilience of the SD-WANs, the key contributions of this dissertation are summarized as follows.

1. We formulate the resilient controller placement problem in SD-WANs, which is more inclusive and easily adjustable (to include single link failures as well) compared with the existing research work, and it is mainly focused on the resilience against controller node failures. The proposed mathematical formulation is among the few schemes that take factors such as the capacities of controllers, the traffic loads of switches, and switch-controller and inter-controller propagation latencies into account simultaneously. We utilize the umbrella term

*resilient controller placement* to refer to our proposed controller placement problem since it covers more than one resilience discipline. Usually, improving one resilience discipline impacts another one. In particular, by assigning more than one controller to a switch, we consider redundancy in our design, which improves both fault tolerance (as a subset of survivability) and reliability (as a subset of dependability). Moreover, QoS measures, i.e., the latency bounds for switch-controller communication and inter-controller communication ensure that our proposed resilient controller placement satisfies SLAs, which are of great importance for service providers, and subsequently result in achieving performability as another resilience discipline. Our proposed resilient controller placement offers more flexibility for the design of an SDN-based network since it is independent from the master controller selection process (which gives more freedom to the selection of a master controller, i.e., not only based on the propagation latency). We model the NP-hard *resilient* CPP based on the *clique* concept in graph theory, and this approach is applicable to other similar variants of reliable facility location problems. While both proposed heuristic algorithms provide high-quality solutions (small gap with the optimal value), the second one gives a solution in polynomial time. Furthermore, a detailed result analysis is provided for different real topologies under various parameter settings.

2. The resilient switch reassignment problem (without/with budget constraints) and the incremental controller placement problem, which both stem from the dynamics of the system (e.g., the variations of the traffic loads of switches) for a given controller placement, are formulated. Such problems in the context of SD-WANs have been less explored especially when considering the capacity of controllers, reassignment budget, and resiliency. Our proposed resilient switch reassignment and incremental controller placement scheme assists in having a better load balancing among the controllers while maintaining the resiliency, satisfying the QoS requirements and/or meeting reassignment cost, which subsequently leads to achieving traffic tolerance as another resilience discipline. We provide algorithms with guaranteed bounds by modeling the aforementioned problems as variants of the classical problem of *scheduling on parallel machines* which incorporate *machine eligibility constraints*, *arbitrary processing sets*, and *resiliency*. We conduct an extensive analysis of the results on real WAN topologies to give detailed insights about the problems.

## 1.4 Dissertation Outline

The outline of this dissertation is as follows.

**Chapter 1** contains the background, research objectives, and contributions in this dissertation followed by an overview of the structure of the dissertation.

**Chapter 2** puts focus on the resilient design of the SDN control plane. In this chapter, we formulate the resilient CPP in SD-WANs, model this NP-hard problem based on the clique concept in graph theory, and solve it using our proposed heuristics.

**Chapter 3** addresses the resilient switch reassignment problem and incremental controller placement problem in SD-WANs. It provides approximation algorithms to solve the aforementioned problems efficiently.

**Chapter 4** concludes the dissertation and discusses the potential future work.

**Appendix A** includes our research work on the incremental migration to SDN while enhancing traffic engineering flexibility for handling networks dynamics in the data plane.

# Chapter 2

# Resilient Controller Placement by Design for Software-Defined WANs

## 2.1 Background

The emergence of SDN, as a promising technology, brings substantial benefits such as network programmability, flexible and efficient network management, vendor-independent control interfaces, accelerated innovation, and cost-effective design and maintenance. By decoupling the routing decision making from packet forwarding, all the control functionalities are incorporated into a (logically) centralized entity called a controller. In particular, SD-WANs have made considerable headway in 2016, and Gartner envisioned that about one-third of network operators will deploy the SD-WAN technology by 2020. Service providers such as CenturyLink, EarthLink, and AT&T have already unveiled SD-WAN services [16]. Moreover, B4 [17], a private WAN connecting Google's data centers, is a practical example for one of the first and largest SDN deployments.

However, the great reliance of SDN on a logically centralized control plane has heightened the concerns of research communities and industries about the resilience of the control plane. Although the controller provides flexible and fine-grained resilience management features that contribute to faster and more efficient failure detection and containment in the network, it is the Achilles' heel of SDN resilience. The malfunction of the control plane resulting from natural disasters, malicious attacks or accidental faults/human errors, may have adverse impacts on the correct functioning of the whole system and affect many applications and services. Thus, connecting

the network devices to a single controller may lead to a single point of failure and a performance bottleneck [18].

The reliable design of the control plane is tightly interwoven with the CPP, which determines the number and the location of controllers in a given topology. Resilient controller placement influences almost all of resilience disciplines [3, 5]. A survey on the main research efforts addressing the resilience disciplines in SDN can be found in [5]. Our focus is on incorporating redundancy into network design, which is one of the most commonly used methods to mitigate the impacts of node/link failures [19]. For instance, one of the fault tolerance techniques adopted in B4 is using software replicas (placed on different physical servers) to protect servers and control processes in case of failures.

With regard to the aforementioned issues, each OpenFlow-enabled switch should be connected to multiple controllers [18] [20] to achieve a resilient control plane. To decrease the communication overhead between the switch and its assigned controllers, instead of having simultaneous connections of a switch to multiple controllers, we focus on a master/slave design. It requires each switch to be connected to one primary controller (master) and one or more slave (backup) controllers. The controller placement should satisfy performance requirements such as the maximum allowable latency between a switch and its assigned controllers as well as the inter-controller communication latency for synchronization purposes. Moreover, the capacity limitation of the controllers as well as the traffic loads of switches should be taken into account. Therefore, designing a control plane which is resilient to controller node failures while satisfying the QoS requirements is of great importance. Although assigning multiple controllers to a switch enhances the resilience of the control plane, it increases the incurred cost in terms of the number of required controllers (each controller incurs the cost of deployment, maintenance, etc). Hence, in order to have a cost-effective design, minimizing the number of controllers is crucial for service providers.

The contribution of this chapter is threefold. First, the *Resilient Capacitated Controller Placement Problem* (RCCPP) in SD-WANs has been formulated, which is more inclusive and easily adjustable when compared with the existing research work, and it is mainly focused on the resilience against controller node failures. The proposed formulation is among the few schemes that take factors, such as the capacities of controllers, the traffic loads of switches and switch-controller and inter-controller propagation latencies into account simultaneously. It also offers more flexibility for the design of an SDN-based network since it is tailored for the satisfaction of the

SLAs by the service providers as well as providing a resilient controller placement scenario that is independent from the master controller selection process. Second, to the best of our knowledge, we are the first one to model the NP-hard RCCPP based on the clique concept in graph theory, and this approach is applicable to other similar variants of reliable facility location problems. While both proposed heuristic algorithms provide high-quality solutions (small gap with the optimal value), the second one gives a solution in polynomial time. Finally, a detailed analysis of the RCCPP is provided for different real topologies under various parameter settings.

## 2.2 Related Work

### 2.2.1 Overview of the CPP in SDN

Given a topology, the CPP (first coined by Heller et al. [10]) finds the number and the locations of required controllers while minimizing the cost associated with the controller placement. This cost can be expressed in terms of the number of controllers or the switch-controller communication latency or the synchronization time of controllers, or a combination of more than one of these metrics (as a multi-objective optimization problem). The surveys in [21–23] provide detailed information about the CPP in SDN. Based on the existing research work on the formulation of the CPP and the provided solutions (mainly in the context of SD-WANs), we list the crucial factors for placing the controllers in an SDN-enabled network as follows.

- **Switch-controller latency**: This is the first and most significant factor for controller placement. Flow setup latency for an unmatched flow in each of the switches is composed of transmission latency, processing latency and propagation latency [24] (as the main contributor to the switch-controller latency in SD-WANs [10, 25]). Long propagation latency between a switch and its assigned controller can adversely affect the capability of the controller to respond to network events in a timely manner and can decrease the communication reliability [14]. Thus, almost all of the research work on the CPP aims to minimize this latency [10, 26–34] or to keep it below a certain threshold [14, 35–41].

- **Inter-controller latency**: This latency is also of great importance, especially for synchronization purposes in case of having multiple controllers assigned to a switch or for inter-domain controller communications [42]. In particular, large

SDN-based networks function according to a global network view that is logically centralized. However, to achieve resiliency and scalability goals, the control state and logic must be physically distributed. Regardless of the methods employed to manage the state consistency of the network, the connectivity among the controllers determines the maximum time required to update information among them [14]. Examples of considering this type of latency can be found in [14, 28, 30, 37, 43]. Similar to the switch-controller latency, it should be minimized or bounded.

- **Controller capacity**: Due to resource constraints (i.e., CPU, memory, and access bandwidth), each controller can only handle a limited number of requests per second. An overloaded controller would have a higher probability of failure [25]. This causes the processing latency to increase, which subsequently affects the switch-controller latency. The capacity of a controller is usually defined as the number of flow setup requests (packets) per second that it can handle [44, 45]. The research work in [24, 25, 29, 30, 39, 45, 46] provides examples of the *capacitated* CPP. It should be noted that load balancing among the controllers does not necessarily correspond to a *capacitated* CPP. For instance, the authors in [28] minimized the *load imbalance*, i.e., the difference between the maximum and minimum number of switches connected to the controllers for improving the load balance among the controllers. Other work such as [47] defined the load on a controller as the number of switches it can manage which ignores the non-uniform traffic loads of switches. No assumption was made to consider the real capacity of the controllers in the aforementioned papers. *A more exact way of defining the load on a controller is the number of requests per second incurred by its connected switches.*

- **Traffic loads of switches**: In a practical SDN controller placement design, the traffic loads of switches should be taken into account to avoid network congestion and the overload of the controllers. This load can be based on the worst/average-case load of switches as in [24, 29, 30, 45] or time-varying traffic loads of switches [35, 39]. It should be noted that the *traffic load of a switch is mainly the number of flow setup requests generated from that switch.*

- **Scalability**: While some of the solutions proposed for the CPP, such as [14, 24], deal with small to medium-scale networks, others are helpful for large-

scale implementations (e.g., [28, 39, 48]). In SD-WANs, with a large number of switches and high traffic volume, controller placement has a great impact on the performance of the system [39].

- **Resilience**: The resilience of the control plane plays a significant role in the sustainability of the entire SDN-based network. Disruptive events may decompose the network and isolate the switches from their assigned controllers [28]. *Resilient* CPP is a variant of the CPP in SDN, which emphasizes the optimization of different reliability aspects of the control plane (examples of existing research are [30, 46, 49–53]). Enhancing the fault tolerance (by assigning more than one controller to a switch) while minimizing the number of required controllers or expected control path loss (i.e., the number of broken control paths resulted from network failures [49]) exemplifies such reliability goals.

It should be noted that there are trade-offs and interdependencies among the aforementioned factors. Therefore, no single best placement strategy exists and the decision makers need to seek a balanced trade-off for a certain use case [28]. Moreover, there exist some overlaps between the CPP and the research on middlebox deployment (such as [54]) and placement of Virtualized Network Function Managers (VNFMs) in virtualized and software-defined networks. However, the CPP differs from such problems in terms of both scalability and dynamics of the system [55]. In addition, [54] does not require reliability and inter-middlebox latency upper bound, both of which are important in the *resilient* CPP. Hence, its offered solution is not applicable to the *resilient* CPP without relaxing the aforementioned key constraints. In the following, we provide an overview of the main existing research work on the *resilient* CPP and highlight its contributions and limitations considering the aforementioned factors.

### 2.2.2 Existing Work on the Resilient CPP

A cloud-based Byzantine Fault Tolerant (BFT) SDN architecture was proposed in [56]. The problem of controller assignment in fault-tolerant SDN was defined as allocating the minimum number of controllers to switches such that BFT protocols can be employed to enhance the resilience of the control plane. However, the capacities of the controllers were uniform and no assumption was made regarding the location of the controllers as well as the controller-switch propagation latencies. The unavailability of a controller to its connected switches may result from single/multiple

switch/link failures on the south-bound connection or the failure of the controller node itself. While having a main connection from the switch to its assigned controller along with multiple auxiliary connections [20] is beneficial for the former case, controller replication is helpful regarding the latter case. Vizarreta et al. [49] proposed two resilient controller placement strategies for tolerating single link and node failures although they did not take the capacity of the controllers into account. While the first strategy involved connecting each switch to its assigned controller through two disjoint paths, the second one required that each switch is connected to two controllers via two disjoint paths. The performance of their solution was evaluated using the expected control path loss and the average control path availability. Also, Gurobi solver was utilized to find the optimal placement for their proposed models.

To achieve a high south-bound reliability, a *resilient* CPP was introduced in [51]. In such a design, each switch is required to satisfy a reliability constraint in a way that the probability of having at least an operational path to its assigned controller(s) is higher than a given threshold. The authors proposed a heuristic to solve the problem. However, the communication/synchronization cost among the controllers was not taken into account. Zhong et al. [38] defined two reliability metrics for the control network based on the average number of disconnected switches resulting from a single physical link failure. Moreover, a heuristic algorithm was proposed to find a min-cover (which was determined by finding the neighborhood of each switch considering the latency bound between the switch and its assigned controller while minimizing the number of controllers) with most reliability. Beheshti and Zhang [57] presented two controller placement algorithms to maximize the controller-switch connection resilience. In addition, by considering both distance and resiliency factors, they proposed an algorithm that constructed a routing tree for the control traffic, which led to short distance as well as high resiliency in the connection between the switches and the controller. Using a similar resilience objective, [58] provided a solution that leverages a min-cut-based graph partitioning algorithm to select the subset of switches for connecting to specific controllers while focusing on switch and link failures.

A cause-based reliability analysis model was proposed in [59] to minimize the expected percentage of control path loss whereas different heuristic algorithms (greedy, simulated annealing and random placements) were evaluated for the same objective in [50]. Guo and Bhattacharya [60] investigated the *resilient* CPP using interdependent network analysis. In particular, an interdependence graph between switch-switch

network (for data forwarding) and controller-switch network (for network control), as two main components of an SDN-based network, was defined. Then, a cascading failure analysis was performed on the interdependence graph. They solved the problem using a greedy optimization method and partitioning scheme for different types of network topologies. To maximize the switch-controller connectivity while satisfying controller capacity constraints in SD-WANs, a solution for the CPP was proposed in [45] along with two failover mechanisms. However, no assumption was made about the switch-controller and inter-controller latencies.

The Pareto-Optimal Controller Placement (POCO) framework in [61–63] was extended in [28] (using the Pareto simulated annealing heuristic) to include large-scale networks. Given the number of controllers, their solution gives Pareto-optimal placements to minimize different objectives, including switch-controller latency, inter-controller latency, load imbalance, and the maximum number of disconnected switches when a node/link failure happens (without incorporating the capacities of the controllers). The research work in [29,37] has similar objective functions while considering the controller failure probabilities and minimizing the total cost including the cost of deployment and expected failure cost.

Alshamrani et al. [52] proposed a model for controller placement to optimize fault tolerance. They considered two metrics, namely the worst-case latency to the $i$-th nearest controller and $\alpha$-adjusted average-case/worst-case latency by incorporating the rate of failure ($\alpha$). The idea behind choosing the first metric was to start with an optimal placement and then remove a subset of controllers to investigate the impact of failure on the average-case and worst-case latencies. Regarding the second metric, they measured the latency performance with regard to the failure of any potential subset of a given set of controllers. However, they could not evaluate their schemes on large topologies due to computational constraints. In [53], the authors modeled the reliability between the controller and switches as the reliability problem of source-to-all-terminals in a network. The shortest paths among the nodes in the network were calculated with regard to the defined reliability metric and the proposed methodology was implemented using ns-3.26, OpenFlow 1.3, and NetAnim.

Among the most recent work is Capacitated Next Controller Placement (CNCP) [30] (a shorter version can be found in [46]), which proposed a *resilient* and capacitated controller placement strategy while considering controller failures. Given a budget in terms of the number of controllers and the inter-controller latency threshold, a switch is assigned to a number of reference controllers and the objective is to

minimize the maximum worst-case latency in case of controller failures. The authors also proposed a simulated annealing heuristic to solve the problem. Although CNCP has the most similarity to our problem ($Q$ parameter corresponds to our parameter $r$), the main differences in terms of formulation and the offered solution are as follows. First, CNCP assumes a given number of controllers as the input of the problem formulation whereas the number of controllers is our objective function. Second, CNCP requires each switch to be connected to its nearest controller as its primary (master) controller; however, this is not necessarily captured in our formulation, which gives more freedom to the selection of a master controller (not only based on the propagation latency). We believe such a selection criterion should be independent from the CPP to have more flexibility in design. Furthermore, connecting switches to their nearest controller(s) may result in more load imbalance among the controllers. We will show later that our formulation achieves similar results, usually with a lower load imbalance, while it is simple, more inclusive (it can easily be converted to the case where the minimization of the total worst-case latencies between the switches and their assigned controllers or between the controllers themselves is required) and easily extendable (especially to include single link failures). Finally, the last difference is that we have proposed a polynomial-time algorithm, which takes into account the structure of the problem using cliques, to solve this NP-hard problem heuristically for large-scale topologies. However, the efficiency of the simulated annealing algorithm for CNCP in terms of both solution quality and time complexity should be investigated for large-scale topologies as it does not guarantee any polynomial-time solution (only a limited evaluation for one medium-size topology was provided in this work and the main focus was on using a commercial optimization solver to acquire a solution).

Among all of the aforementioned research work on the *resilient* CPP, the capacity of controllers and the load of switches were only considered in [29, 30, 45]. Also, some research work did not take the inter-controller latency into account and no comparison was made with the optimal solution when heuristics were proposed. Moreover, since the CPP is NP-hard [10], most of the proposed solutions for *resilient* CPP are not appropriate for large-scale networks due to the enormous time required to search the solution space, especially when multiple factors are considered simultaneously. Therefore, a formulation of the *resilient* CPP which incorporates all of the important factors while being easily adaptable is of great significance.

## 2.3 System Model and Problem Formulation

### 2.3.1 Preliminaries

The CPP and its *resilient* form are variants of the Facility Location Problem (FLP) [28, 64], which is an NP-hard problem [10]. In the discrete form of such a problem, we have a finite set of users with their associated demands for service and a finite set of potential locations to place the facilities. In an SD-WAN, the controllers play the role of the facilities and switches are the customers/clients. While many of the publications on the *resilient* CPP (e.g., [37, 51]) have investigated the uncapacitated version, we focus on the capacitated version which is a better representation of many real-life applications, including the *resilient* CPP in SDN. Generally, to seek the solution for such problems, two types of decisions should be made. Location decisions determine where to place the controllers while the assignment decisions involve how to allocate the established controllers to the switches.

### 2.3.2 Our Assumptions

#### Single and multi-controller node failures

It should be noted that we consider controller node failures in contrast with controller site failures. The latter applies to a more severe impact resulting from a disaster/attack that completely destroys the data center where the controller is located rather than the controller itself, and it is less frequent compared with the former one. To improve the resilience of the control plane, multiple controllers should be assigned to a switch. We focus on a master/slave model in which we have a primary (master) controller (with full control over the switch) assigned to a switch along with one or more backup controllers (with read-only access to the switch). While having one backup controller results in tolerance to single controller failures, having more than one backup controller (as a design parameter indicated by the network designer) leads to handling multiple controller failures. This planning ahead for controller failures (rather than manual and administrative intervention) is in line with some of the existing work on *resilient* CPP such as [29, 30, 45]. Although a simple fail-over mechanism which involves defining a list of controllers for a switch was proposed in OpenFlow v0.9, the controller role change mechanism to support multiple controllers for fail-over and load balancing purposes is included in OpenFlow v1.2 and the later

versions (more information can be found in [20]). The assigned controllers to the switch organize the management of the switch among themselves and they make the decision to choose the master controller. Such a coordination mechanism is also needed for distributed controllers (as in [42]) and it is out of the scope this chapter. The OpenFlow specification [20] gives freedom to the network designers to utilize a synchronization/coordination mechanism/protocol of their choice.

**Objective of the optimization problem**

Due to the incurred cost of having multiple controllers assigned to a switch for increasing reliability, a preferable solution to the *resilient* CPP for service providers is the one that is more cost-effective (i.e., minimizes the total number of controllers). However, one can minimize the average or the worst-case switch-controller latency in a multi-objective optimization problem or when a budget is given in terms of the number of controllers (e.g., [30]) similar to $p$-median and $p$-center FLPs. We believe that it is more practical to give the network designer the freedom to choose and tune different QoS parameters (switch-controller latency and inter-controller latency) to minimize the cost of resilient controller placement before the final deployment according to the requirements of a certain network. Thus, by changing the aforementioned parameters for a given topology, the required budget is changed accordingly.

**Bounds for switch-controller and inter-controller latencies**

The interplay between switch-controller and inter-controller latencies has been studied in some of the existing work such as [28]. A group of controllers which are close to each other leads to low inter-controller latencies and high switch-controller latencies whereas spatially distributed controllers result in the opposite case. Connecting switches to their nearest controller(s) may result in load imbalance among the controllers, and subsequently increase latency due to queuing time at some of the controllers [28]. In contrast, considering a threshold for switch-controller and inter-controller latencies can guarantee the latency not to exceed the latency upper bound. This case can be converted to the former case by reducing the threshold as much as possible for a given topology (i.e., more than or equal to the minimum propagation latency between two nodes). Moreover, the threshold-based approach is critical for delay-sensitive applications or for the satisfaction of the SLAs by service providers. Considering the aforementioned issues, in this dissertation, we follow the

latency bound-based approach, which is more inclusive, especially when the number of required controllers needs to be minimized. Since the switch-controller communication is more frequent than the inter-controller interactions, we assume that the former threshold is less than or equal to the latter one. In addition, both switch-controller and inter-controller latencies can be approximated by the propagation latency in SD-WANs (due to the fact that it is the main part of the total latency). Furthermore, since each controller is in charge of managing a subset of all switches and due to having a distributed control plane, to maintain a consistent global view of the network and subsequently to ensure the proper functioning of the network, not only the primary and backup controllers of a switch should be synchronized with each other, but also all the controllers need to communicate with each other [30]. Regardless of the methods utilized to manage the network state consistency, the connectivity among controllers determines the maximum time required to update information among them [14]. Therefore, the inter-controller latency should be embedded into problem formulation.

#### Single link failures

Although our key focus is on controller node failures similar to [30], we show the extension of our problem formulation to include link-disjoint paths between a switch and its assigned controllers.

### 2.3.3 Problem Formulation

The topology of an SD-WAN is represented by a *connected* graph $\boldsymbol{G}(\boldsymbol{V}, \boldsymbol{E})$, where $\boldsymbol{V} = \boldsymbol{S} \cup \boldsymbol{C}$, $\boldsymbol{S}$ is the set of OpenFlow-enabled switches, and $\boldsymbol{C}$ is the set of potential controller locations while $\boldsymbol{E}$ denotes the set of weighted links. The weights of the links are the propagation latencies (shortest path lengths) between the nodes based on their geographical locations. Assuming that the controllers can share the same location with the switches, the potential locations for the controllers are equal to the set of switches (i.e., $\boldsymbol{C} = \boldsymbol{S}$). We define two binary variables, namely $y_j$ and $x_{ij}$ to determine the controller location decisions and the assignments of controllers to the switches, respectively. The RCCPP is defined as follows.

Minimize

$$\sum_{j \in \boldsymbol{C}} y_j, \tag{2.1}$$

subject to

$$y_j \geq x_{ij}, \quad \forall i \in \boldsymbol{S},\ j \in \boldsymbol{C} \tag{2.2}$$

$$\sum_{j \in \boldsymbol{C}} x_{ij} = r, \quad \forall i \in \boldsymbol{S} \tag{2.3}$$

$$\sum_{i \in \boldsymbol{S}} l_i\ x_{ij} \leq u_c, \quad \forall j \in \boldsymbol{C} \tag{2.4}$$

$$d_{ij}\ x_{ij} \leq sc_{\max}, \quad \forall i \in \boldsymbol{S},\ \forall j \in \boldsymbol{C} \tag{2.5}$$

$$d_{j'j''}\ y_{j'}\ y_{j''} \leq cc_{\max}, \quad \forall j',\ j'' \in \boldsymbol{C} \tag{2.6}$$

$$x_{ij}, y_j \in \{0, 1\}, \quad \forall i \in \boldsymbol{S},\ \forall j \in \boldsymbol{C}. \tag{2.7}$$

The constraint in (2.2) prohibits a switch from being assigned to a controller site which is not open while the constraint in (2.3) ensures that each switch is connected to $r > 1$ controllers (if $r = 1$, the formulation corresponds to the *capacitated* CPP). Note that having $r = 2$ emphasizes resilience against single controller node failures while $r > 2$ corresponds to resilience against the multi-controller failure case. The constraint in (2.4) prevents the total incurred load by the switches on a controller from exceeding its capacity ($u_c$ denotes the capacity of a controller and we assume that all controllers have a uniform capacity which is given as one of the input parameters of the problem). The constraint in (2.5) expresses that the propagation latency between a switch and its assigned controllers satisfies the latency bound $sc_{\max}$. Satisfying the maximum allowed latency among the open controllers is ensured by the constraint in (2.6). Finally, (2.7) provides the integrality constraints. Since the constraint

in (2.6) is non-linear, we linearize it by defining a new binary variable $w_{j'j''}$ using the McCormick envelopes [65]), which is given by

$$w_{j'j''} = y_{j'} y_{j''}, \tag{2.8}$$

and subsequently replacing it with the following constraints

$$d_{j'j''}\ w_{j'j''} \leq cc_{\max}, \quad \forall j',\ j'' \in \boldsymbol{C} \tag{2.9}$$

$$w_{j'j''} \leq y_{j'}, \quad \forall j',\ j'' \in \boldsymbol{C} \tag{2.10}$$

$$w_{j'j''} \leq y_{j''}, \quad \forall j',\ j'' \in \boldsymbol{C} \tag{2.11}$$

$$w_{j'j''} \geq y_{j'} + y_{j''} - 1, \quad \forall j',\ j'' \in \boldsymbol{C} \tag{2.12}$$

$$w_{j'j''} \in \{0, 1\}, \quad \forall j',\ j'' \in \boldsymbol{C}. \tag{2.13}$$

The above problem formulation can be extended by adding the following constraint to include the protection against single link failures. $DP(i, j', j'')$ is 1 if all the control paths of switch $i$ (i.e., the paths to its assigned two controller $j'$ and $j''$) are link-disjoint, and 0 otherwise.

$$x_{ij'} x_{ij''} \leq DP(i, j', j''), \quad \forall i \in \boldsymbol{S}\ \forall j',\ j'' \in \boldsymbol{C}. \tag{2.14}$$

The constraint in (2.14) can be linearized by introducing a three-indexed binary variable $z_{ij'j''}$ using the McCormick envelopes similar to the constraint in (2.6). $DP$ denotes a function that determines whether or not all the control paths for a switch are link-disjoint. The input of this function is switch $i$ and two potential controller locations ($j'$ and $j''$). Therefore, its output is equal to 1 if the control paths of any two controllers (deployed at nodes $j'$ and $j''$) of assigned controllers to switch $i$ are link-disjoint. Note that the shortest paths between a node and all other nodes of $\boldsymbol{G}$ are the input of the optimization problem. It should be noted that the values of

both $cc_{\max}$ and $sc_{\max}$ are indicated by a fraction of the graph diameter for consistency across various topologies (similar to some of the existing work such as [28] [30]). We denote the diameter of the given WAN topology $\boldsymbol{G}$ by $D_G$ (the length of the longest shortest path) and we indicate the minimum shortest path length in $\boldsymbol{G}$ by $D_{\min}$. We assume the following holds.

$$D_{\min} \leq sc_{\max} \leq cc_{\max} \leq D_G. \tag{2.15}$$

Table 2.1 summarizes the notation used in the formulation of RCCPP and the proposed algorithms in Section 2.4.

Table 2.1: Notation used in the problem formulation and proposed algorithms

| Symbol | Definition |
|---|---|
| $\boldsymbol{S}$ | Set of OpenFlow-enabled switches |
| $\boldsymbol{C}$ | Set of potential controller locations |
| $\boldsymbol{N}$ | Network size, i.e., $\lvert\boldsymbol{S}\rvert$ |
| $\boldsymbol{G}$ | Topology of an SD-WAN |
| $\boldsymbol{G_o}$ | Overlay graph |
| $\boldsymbol{G_p}$ | Pruned overlay graph |
| $\boldsymbol{M}$ | Set of all maximal cliques of $\boldsymbol{G_p}$ |
| $\boldsymbol{A}$ | Set of all $r$-cliques and $(r+1)$-cliques of $\boldsymbol{G_p}$ |
| $\boldsymbol{A_i}$ | A subset of $\boldsymbol{A}$ that includes switch $i$ |
| $\omega(\boldsymbol{G_p})$ | Clique number of $\boldsymbol{G_p}$ |
| $\boldsymbol{F}$ | Set of all found feasible solutions for RCCPP-AMC |
| $LB$ | Lower bound of the number of controllers |
| $sc_{\max}$ | Latency bound between a switch and its assigned controllers |
| $cc_{\max}$ | Inter-controller latency bound |
| $u_c$ | The capacity of a controller |
| $l_i$ | The traffic load of switch $i$ |
| $r$ | Number of controllers to serve a given switch (resilience parameter) |
| $d_{ij}$ | Minimum propagation latency between node $i$ and node $j$ |
| $y_j$ | 1 if node $j$ is selected to deploy a controller, and 0 otherwise |
| $x_{ij}$ | 1 if switch $i$ is connected to the controller at node $j$, and 0 otherwise |
| $w_{j'j''}$ | 1 if two controllers are at nodes $j'$ and $j''$, respectively, and 0 otherwise |
| $z_{ij'j''}$ | 1 if switch $i$ is connected to the controllers $j'$ and $j''$, and 0 otherwise |
| $DP$ | 1 if all the control paths of a switch are link-disjoint, and 0 otherwise |

## 2.4 Proposed Solution

In this section, we elaborate our idea to solve the formulated optimization problem in Section 2.3 based on the clique concept in graph theory by introducing two heuristic

algorithms. Then, a case study is provided to delineate the proposed algorithms.

### 2.4.1 Clique Graphs and their Relevance to Our Problem

We define a complete graph (denoted by $\boldsymbol{G_o}$) of the physical network topology as an overlay, in which the nodes correspond to the switches and/or controllers and the weights of the links correspond to the shortest path lengths between each pair of nodes. Then, we prune $\boldsymbol{G_o}$ by removing the links which do not satisfy the latency bound $cc_{\max}$ and we call the resultant graph $\boldsymbol{G_p}$. In this graph, the existence of a link between each pair of nodes means that these nodes can be in the set of controllers in a potential solution. By studying the structure of the optimal solution to the formulated problem in Section 2.3, we observe that the set of controllers in the solution as well as each switch and its assigned controllers is a clique of $\boldsymbol{G_p}$. A clique [66] is defined as a complete subgraph of an undirected graph. In particular, the inter-controller latency in the constraint (2.6) implies that the set of controllers in a solution must be a subset of one of the maximal cliques[1] of $\boldsymbol{G_p}$. All of the controllers need to be directly connected to each other, and hence they must form a clique, which is a subset of a maximal clique of $\boldsymbol{G_p}$. Moreover, since a switch needs to be directly connected to all of its assigned $r$ controllers and such controllers themselves are required to interact with each other (and thus, each pair of the $r$ controllers must be adjacent in $\boldsymbol{G_p}$), the switch and its associated controllers form a complete subgraph, i.e., a clique of $\boldsymbol{G_p}$. Therefore, possible controller-switch assignments are the $r$-cliques and $(r+1)$-cliques (if any) of $\boldsymbol{G_p}$. Cliques of size $r$ correspond to the case where one of the potential controllers of the switch is co-located with it while $(r+1)$-cliques indicate that none of the assigned controllers to a switch is co-located with it. Based on all these observations and insights of the optimal solution, we have developed two heuristic algorithms to solve the problem, the description of which is provided in Section 2.4.2.

**Lower bound and upper bound of the objective function value**: Due to the fact that the controllers in a feasible/optimal solution form a clique which is a subset of one of the maximal cliques, the upper bound of the number of controllers (i.e., the value of the objective function) is equal to the clique number (i.e., size of the maximum clique[2]) of $\boldsymbol{G_p}$ denoted by $\omega(\boldsymbol{G_p})$. Moreover, the number of controllers in an

[1] A clique is maximal if it cannot be extended (turned into a larger clique) by adding more adjacent vertices to it [66].

[2] A clique is maximum, if there is no other clique of larger size in the graph [66].

**Algorithm 1** General algorithmic framework

1: Input: $\boldsymbol{G}$, $cc_{\max}$, $sc_{\max}$, $r$, switch loads, controller's capacity ($u_c$), shortest paths matrix.
2: Output: controller locations and controller-switch assignments or infeasible state.
3: Feasibility-Check ($\boldsymbol{G}$, $cc_{\max}$, $sc_{\max}$).
4: $\boldsymbol{G_o}$ = OverlayGraph ($\boldsymbol{G}$).
5: $\boldsymbol{G_p}$ = Prune ($\boldsymbol{G_o}, cc_{\max}$).
6: Feasibility-Check ($\boldsymbol{G_p}$).
7: $\boldsymbol{A}$ = all $r$-cliques and $(r+1)$-cliques of $\boldsymbol{G_p}$.
8: For each switch $i$, find a subset of $\boldsymbol{A}$ ($\boldsymbol{A_i}$) that includes that switch with regard to the values of $sc_{\max}$ and $cc_{\max}$.
9: Sort the switches with regard to the total number of their associated cliques ascendingly.
10: Feasibility-Check ($\boldsymbol{S}$, $\boldsymbol{A_i}$), $\forall i \in \boldsymbol{S}$.
11: RCCPP-AMC ().
12: RCCPP-SMC ().

optimal solution has to be at least equal to the total traffic loads of switches divided by the capacity of a controller (assuming uniform capacity $u_c$ for all the controllers) as well as having to be greater than or equal to the value of $r$. Hence, the following must hold

$$\left\lceil \max\left(r, \frac{r \times \sum_{i \in S} l_i}{u}\right)\right\rceil \leq y^* \leq \omega(\boldsymbol{G_p}), \tag{2.16}$$

where $y^*$ denotes the optimal (minimum) number of controllers in a solution.

### 2.4.2 Descriptions of the Proposed Algorithms

*Algorithm* 1 serves as a general framework that includes the common steps of our proposed algorithms, namely RCCPP-AMC (RCCPP with All Maximal Cliques) and RCCPP-SMC (RCCPP with at least a Single Maximal Clique). The first step is the feasibility check with regard to (2.15) (which requires constant time). Building the overlay graph $\boldsymbol{G_o}$ by *OverlayGraph* ($\boldsymbol{G}$) has the time complexity of $O(N^2)$ (where $N = |\boldsymbol{S}|$). The process of pruning the complete graph $\boldsymbol{G_o}$ is of $O(N^2)$ complexity since it involves checking all the edges. Then, a feasibility check for $\boldsymbol{G_p}$ with regard to the chosen value of $cc_{\max}$ is performed in step 6. If $\boldsymbol{G_p}$ is a disconnected graph, the problem is infeasible (checking the connectivity of $\boldsymbol{G_p}$ takes $O(N^2)$ time using

**Algorithm 2** RCCPP-AMC

1: $\boldsymbol{M}$ = All-Maximal-Cliques ($\boldsymbol{G_p}$).
2: Feasibility-Check ($\boldsymbol{M}, LB$).
3: $\boldsymbol{F} = \varnothing$.
4: **for** $m \in \boldsymbol{M}$ **do**
5: Find a feasible solution (if any) and add it to $\boldsymbol{F}$.
6: **end for**
7: **if** $\boldsymbol{F}! = \varnothing$ **then**
8: Output the best solution.
9: **else**
10: The problem is infeasible.
11: **end if**

BFS or DFS). As shown in step 7 of *Algorithm* 1, to identify the possible controller-switch assignments, we find the sets of all $r$-cliques and $(r+1)$-cliques (if any) of $\boldsymbol{G_p}$. Given the fixed value of $r$ in RCCPP, finding cliques of size $r$ and $r+1$ takes $O(N^r)$ and $O(N^{r+1})$ time, respectively. In particular, for finding the $r$-cliques, $\binom{N}{r}$ subgraphs (with $r$ vertices) should be checked and since the subgraphs of interest must be complete, the presence of at most $r(r-1)/2$ edges in each subgraph must be checked. Thus, the worst-case time complexity of this operation is $O(r^2 N^r)$, which is converted to the polynomial form $O(N^r)$, thanks to the fixed value of $r$ in our problem. A similar explanation applies to finding cliques of size $r+1$. However, more efficient algorithms for finding the cliques of fixed size have been found in [67]. Then, for each switch, we define the set of all cliques that include switch $i$ ($\boldsymbol{A_i}$) according to the following two cases:

1. $sc_{\max} = cc_{\max}$: $\boldsymbol{A_i}$ includes all $r$-cliques as well as $(r+1)$-cliques (if $r < \omega(\boldsymbol{G_p})$) that contain switch $i$.

2. $sc_{\max} < cc_{\max}$: $\boldsymbol{A_i}$ includes a number of $r$-cliques and/or $(r+1)$-cliques such that the weight of all incident links to switch $i$ in each of such cliques is less than or equal to the value of $sc_{\max}$.

We sort the switches according to the size of their associated $\boldsymbol{A_i}$ (i.e., the number of possible controller assignments for each switch $i$) in an increasing order ($O(N \log N)$ time). This means that the switches with fewer possible sets of assignments are handled first. If there is at least one switch $i$ with $|\boldsymbol{A_i}| = 0$, the problem becomes infeasible (step 10 of *Algorithm* 1).

**RCCPP-AMC**: As shown in *Algorithm* 2, the set of all maximal cliques of $\boldsymbol{G_p}$ (denoted by $\boldsymbol{M}$) is computed and the maximal cliques whose number of nodes is less than the lower bound calculated in (2.16) are excluded from $\boldsymbol{M}$. Thus, if $\boldsymbol{M}$ becomes empty after checking the aforementioned condition, the problem becomes infeasible (step 2). Then, for each of the remaining maximal cliques, it is assumed that all the nodes in that maximal clique are open and the algorithm proceeds with finding the assignments for each switch, i.e., a feasible solution is found if there is any (step 5). In particular, to choose among the cliques of a switch in $\boldsymbol{A_i}$, we first leave out all the $r$-cliques and $(r+1)$-cliques whose potential controller nodes are not a subset of the currently chosen maximal clique $m$. In addition, all the cliques that have at least a controller node such that its remaining capacity is less than the traffic load of switch $i$, are excluded from $\boldsymbol{A_i}$. Afterward, if there is any clique whose controllers have been used already (i.e., their remaining capacity is less than the initial capacity), that clique is chosen as the assignment for switch $i$. Otherwise, we rank the cliques based on the number of existing used controllers in them, and then we choose the clique with the highest rank as the assignment for switch $i$. This results in the reuse of already assigned controllers as much as possible. If a clique is found, the controllers in this clique are assigned to switch $i$. Once we are done with the assignments for all switches, if there is any controller in the chosen maximal clique $m$ that is not involved in any of the controller-switch assignments, it is removed from the set of open controllers in the found solution. This solution is added to the list of found solutions. Finally, if more than one feasible solution is found, the best one (with the least number of controllers) is selected (if there exists only one feasible solution, it will be chosen as the best solution), otherwise the problem is infeasible (steps 7–11). RCCPP-AMC has a high chance of escaping the local optima by finding all the maximal cliques to produce multiple feasible solutions of good quality and then choosing the solution with the minimum number of controllers.

**Time complexity of RCCPP-AMC**: Finding all maximal cliques has $O(3^{N/3})$ worst-case running time, since any arbitrary graph with $N$ vertices has at most $3^{N/3}$ maximal cliques [68, 69]. However, it is possible to list all of the maximal cliques in polynomial or even in linear time for special families of graphs [70, 71]. For instance, in our problem, if $cc_{\max} = D_G$, $\boldsymbol{G_p}$ is a complete graph which is its own maximal clique. Similar observations are true for the cases where $\boldsymbol{G_p}$ is a planar graph [70] or a sparse graph [72]. The running time of *Feasibility-Check* $(\boldsymbol{M}, LB)$ is dominated by $O(3^{N/3})$ to exclude the maximal cliques that do not satisfy the total traffic loads

**Algorithm 3** RCCPP-SMC

```
1: F = ∅.
2: for a ∈ A do
3:     Construct a single maximal clique m using clique a.
4:     if visited (m) or Infeasible (m, LB) then
5:         Goto 2.
6:     end if
7:     Find a solution with regard to the sorted list of switches.
8:     if a feasible solution is found then
9:         Add it to F.
10:    end if
11: end for
12: if F! = ∅ then
13:    Output the best solution.
14: else
15:    The problem is infeasible.
16: end if
```

of switches. Note that finding assignments for a switch has running time of at most $O(N^{r+1})$ (the maximum number of $(r+1)$-cliques). Finally, the overall worst-case time complexity of the algorithm is $O(3^{N/3})$.

**RCCPP-SMC (a polynomial-time algorithm)**: RCCPP-AMC (shown in *Algorithm* 3) finds all maximal cliques of $\boldsymbol{G_p}$ and chooses the solution with the best quality among all the found feasible solutions (if any). However, it has an exponential time complexity which is not desirable in practical settings and for large-scale networks. Therefore, we propose a polynomial-time algorithm, RCCPP-SMC, which also gives us high-quality solutions (while not affecting the quality of many solutions found by RCCPP-AMC to a great degree as shown in Section 2.5). The operational difference of this algorithm comparing with RCCPP-AMC is as follows. As shown in *Algorithm* 3, we compute a single maximal clique based on an element (a clique) of $\boldsymbol{A}$ using the algorithm in [73]. That is, we begin with a single clique in $\boldsymbol{G_p}$ and try to add the other nodes of the graph to this clique one by one. A node is added if it is a neighbor of all the nodes inside the clique. The aforementioned algorithm for finding a single maximal clique has time complexity $O(N^2)$. If the created maximal clique has not been used before (not visited) and the number of nodes in this maximal clique is greater than or equal to the lower bound calculated in (2.16), the algorithm proceeds to the next step. Otherwise, another element of $\boldsymbol{A}$ (if any) is chosen. The constructed maximal clique serves as the set of potential locations for controllers. Then, we as-

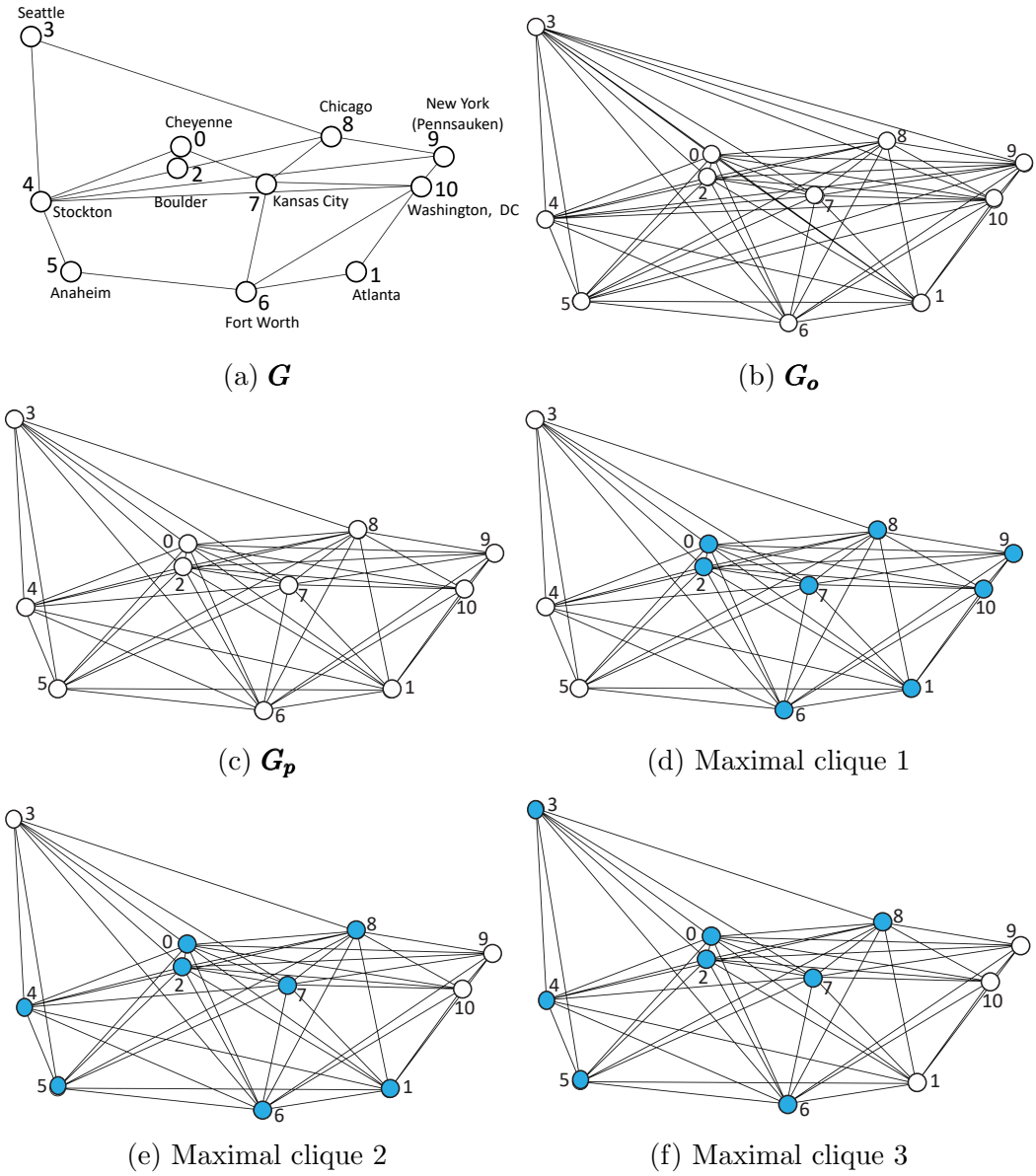
sign the controllers from this set to other switches following the same approach in RCCPP-AMC. If a feasible solution is found, it is added to the set $\boldsymbol{F}$; otherwise we choose another element of $\boldsymbol{A}$ (if any) and repeat steps 3–10. Finally, if $\boldsymbol{F}$ is not empty, the best feasible solution (with the least number of controllers) is selected, otherwise the problem is infeasible. The worst-case time complexity of RCCPP-SMC is $O(N^{2r+3})$. This is due to the fact that the maximum number of iterations of the *for loop* is $O(N^{r+1})$ (the number of elements in $\boldsymbol{A}$) and finding the assignments for all the switches in step 7 has the highest time complexity ($O(N) \times O(N^{r+1})$) among other steps inside the loop.

### 2.4.3 Case Study

To illustrate the effectiveness of the proposed algorithm, we study an example for the Sprint topology. We set the input parameters as follows: $cc_{\max} = 0.8D_G$, $sc_{\max} = 0.4D_G$, $r = 2$, $u_c = 2000$ $k$req/s (controller capacity) and $l_s = 200$ $k$req/s (uniform switch traffic loads). The original Sprint topology $\boldsymbol{G}$, $\boldsymbol{G_o}$, $\boldsymbol{G_p}$, and the set of all three maximal cliques of $\boldsymbol{G_p}$ are shown in Figure 2.1. The lower bound of the number of controllers in the optimal solution is 3 whereas the upper bound is 8 (i.e., the clique number of $\boldsymbol{G_p}$). In this example, the obtained solutions by both RCCPP-AMC and RCCPP-SMC are optimal. The set of open controllers in both solutions is $\{1, 4, 5, 6, 7\}$, which is a subset of maximal clique 2 (Figure 2.1e). The difference between RCCPP-AMC and RCCPP-SMC is that the former finds all of the 3 maximal cliques and finds the best solution among the ones produced by each of these maximal cliques while the latter only finds a single maximal clique from the 2-clique $\{3, 4\}$ for switch 3 (which is the first switch in the sorted list of the switches). Figure 2.2 depicts the controller-switch assignments in the solution. More specifically, these assignments are the subsets of the 2-cliques and 3-cliques of $\boldsymbol{G_p}$. The switch nodes are marked in blue. The controller nodes not co-located with the switch they serve are marked with red color while the ones co-located with the switch they serve are highlighted by orange color.

## 2.5 Performance Evaluation

In this section, we first provide a detailed description of our experiment setup and then we assess the performance of our proposed solutions with regard to different




(a) $\boldsymbol{G}$ (b) $\boldsymbol{G_o}$

(c) $\boldsymbol{G_p}$ (d) Maximal clique 1

(e) Maximal clique 2 (f) Maximal clique 3

Figure 2.1: Sprint topology and its corresponding constructed graphs as well as the maximal cliques of $\boldsymbol{G_p}$.

metrics and parameters.

### 2.5.1 Experiment Setup

We conducted our experiments on about 40 WAN topologies from Internet Topology Zoo (ITZ) [74], which is a publicly available dataset and it has been used by much of the research work on SDN controller placement problems such as [10, 28, 51]. Such

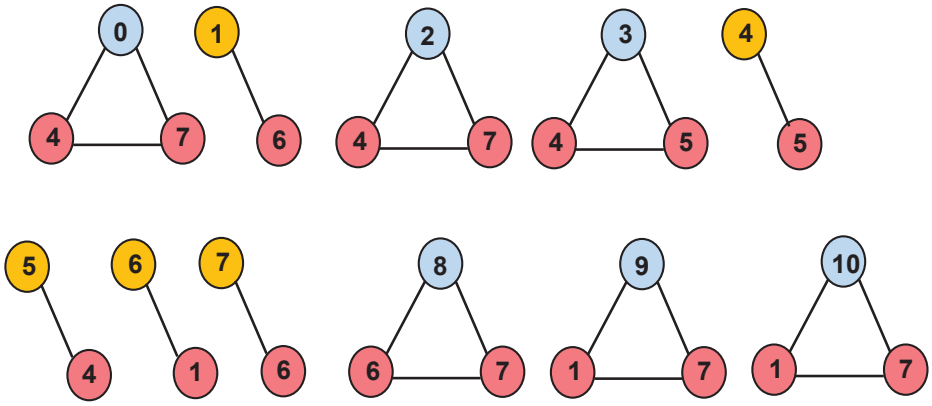



Figure 2.2: Controller-switch assignments for the Sprint topology.

network maps are of great importance in the optimization of network design and they represent the level at which the resilience and redundancy are highly likely to be considered. Moreover, the aforementioned dataset contains a broad range of topologies spanning over different geographical areas (ranging from regional/state networks to the continental ones). For the ease of analysis and presentation, the chosen topologies were classified according to their sizes (i.e., the number of nodes $N$). Four groups were defined and labeled as follows. Groups 1 ("small-size"), 2 ("medium-size"), 3 ("large-size"), and 4 ("very large-size") include the topologies with $N < 20$, $20 \leq N < 50$, $50 \leq N < 100$, and $N \geq 100$, respectively. As representatives for each group, we chose multiple graphs that cover different types of topologies (i.e., mesh, linear, ring and hub-and-spoke). Note that we chose the most recent version of a topology if there were more than one version. The following shows the summary of the steps taken to conduct the experiments.

### Pre-processing

For this step, we applied a similar approach as in [10, 28]. Multi-graphs were converted to simple graphs (the parallel edges do not affect the propagation latencies) and nodes with missing location information (i.e., latitude and longitude) were removed from the graph. The number of node removals was negligible with regard to the topology size (e.g., for TATA, 2 out of 145 nodes were removed). If the graph was disconnected, the largest connected component was taken into account. We assigned weights to the edges from the calculated propagation latencies (based on the

Table 2.2: Parameters and their associated values

| **Parameter** | **Value** |
|---|---|
| Controller capacities ($u_c$) | $\{2000, 5000, 10000\}$ $k$req/s |
| Homogeneous traffic loads for the switches ($l_s$) | 200 $k$req/s |
| Heterogeneous traffic loads for the switches ($l_s$) | $(0, 400]$ $k$req/s |
| Number of experiments | 50 |
| Resilience level (r) | $\{1, 2, 3\}$ |
| $cc_{\max}$ | $\{1, 0.8, 0.6\}D_G$ |
| $sc_{\max}$ | $\{1, 0.8, 0.6, 0.4\}D_G$ |

geodesic distance). Also, the shortest path lengths between nodes were calculated using Dijkstra's algorithm.

### Parameter settings

Table 2.2 shows the assigned values to different parameters of the problem in our experiments. Uniform capacities were associated to the potential controllers while both homogeneous and heterogeneous traffic loads for the switches were considered. For the heterogeneous case, the traffic loads of switches (as integer numbers) were uniformly distributed in $(0, 400]$ $k$req/s. All the applied values were based on prior studies on the *capacitated* CPP [25,39,45] as well as the research conducted on the performance of SDN controllers [75, 76]. Considering the heterogeneous load for the switches, 50 independent experiments were conducted to obtain the results. The resilience level $r$ was set to 1, which indicates the *capacitated* CPP (i.e., no resilience), and 2 and 3 to specify RCCPP. Note that $r = 2$ and $r = 3$ indicate the resilience against single controller node failures and dual-controller node failures, respectively. The values for $cc_{\max}$ and $sc_{\max}$ were chosen as a percentage of $D_G$ (the largest possible propagation latency for a given topology) for keeping the consistency across various topologies, which is also in line with most of the existing work such as [28,30]. However, network designers may choose specific values according to the QoS requirements of a specific network. Such specific values can still be represented as a percentage of $D_G$ and it is clear that for values higher than $D_G$, the controller placement is not affected by the shortest path lengths in the topology graph.

**Obtaining the results**

The Python interface of the Gurobi optimization software (version 6.5.2) [77] was used to obtain the optimal solutions. Furthermore, Python code was developed to solve RCCPP based on the proposed algorithms. All the experiments were carried out on an Intel(R) Core(TM) i7-3770 CPU @3.40GHz and 32GB RAM with Windows 10 Pro (64-bit) installed. For each topology, the results shed light on the feasibility of using certain switch-controller and inter-controller latency values to satisfy a resilience level while minimizing the number of controllers.

In the following, we first compare our scheme with CNCP [30], which is the most relevant existing work on the *resilient* CPP to ours. Then, we make a comparison between our proposed algorithms with regard to their solution quality. We also provide a sensitivity analysis by considering the impact of different factors (including topology, switch-controller/inter-controller latency thresholds, and the controller capacity) on the number of controllers (our objective function). The change in the number of controllers may subsequently affect the utilization of the controllers. Furthermore, we discuss the dominant role of $sc_{\max}$ on the infeasibility of the problem instances. Finally, we briefly present the viability of incorporating protection against single-link failures that affect the connectivity between a switch and its assigned controllers.

### 2.5.2 Comparison with CNCP

We delineated the differences between our problem formulation and CNCP in Section 2.2.2. To compare RCCPP with CNCP, the optimal solutions to RCCPP and to CNCP were acquired for small to large topologies. In particular, we set $cc_{\max} = 0.6D_G$ (corresponding to parameter $\gamma$ in CNCP), $r = 2$ (corresponding to parameter $Q$ in CNCP), and $u_c = 2000$ $k$req/s, and we considered a homogeneous traffic load for the switches. Then, we solved RCCPP using the formulation in Section 2.3.3 by setting $sc_{\max} = 0.6D_G$. Next, we decreased the value of $sc_{\max}$ (gradually by steps of 0.05 or 0.1) such that RCCPP was still feasible with the same number of controllers (as the case for $sc_{\max} = 0.6D_G$) and recorded the solutions. Afterwards, CNCP was solved by limiting the number of controllers to the value obtained by RCCPP.

The key differences in the output of the two schemes lie in the location and assignment decisions, which subsequently affect the load imbalance (i.e., the difference between the load of the controller with the highest remaining capacity and that of the controller with the lowest remaining capacity). For most of the topologies, RCCPP

Table 2.3: Comparison between RCCPP and CNCP in terms of load imbalance

| **Topology** | **Size** | **Number of Controllers** | **Average Controller Utilization** | $\boldsymbol{sc_{\max}}$ | **Load Imbalance (RCCPP)** | **Load Imbalance (CNCP)** |
|---|---|---|---|---|---|---|
| GlobalCenter | 9 | 3 | 60% | $0.59D_G$ | 600 $k$req/s | 800 $k$req/s |
| GridNet | 9 | 3 | 60% | $0.56D_G$ | 1000 $k$req/s | 1200 $k$req/s |
| NAVIGATA | 13 | 3 | 86.67% | $0.52D_G$ | 600 $k$req/s | 800 $k$req/s |
| GoodNet | 17 | 4 | 85% | $0.6D_G$ | 400 $k$req/s | 800 $k$req/s |
| BELNET | 19 | 16 | 95% | $0.538D_G$ | 1000 $k$req/s | 1200 $k$req/s |
| KAREN | 23 | 5 | 92% | $0.5D_G$ | 400 $k$req/s | 600 $k$req/s |
| PSINet | 24 | 5 | 96% | $0.45D_G$ | 200 $k$req/s | 400 $k$req/s |
| UUNET | 42 | 9 | 93.33% | $0.4D_G$ | 400 $k$req/s | 600 $k$req/s |
| GARR | 48 | 10 | 96% | $0.5D_G$ | 200 $k$req/s | 400 $k$req/s |
| DFN | 51 | 11 | 92.73% | $0.5D_G$ | 800 $k$req/s | 1000 $k$req/s |

provides a load imbalance which is equal to or lower than that of CNCP with almost a similar maximum switch-controller latency. A lower load imbalance is desirable due to having a better load distribution among the controllers as well as the decline in the queueing delay of the controllers. Table 2.3 summarizes the results of the topologies for which RCCPP achieves a lower load imbalance than CNCP. This stems from the fact that the switches are not necessarily connected to their nearest controllers in RCCPP. It should be noted that the fifth column of Table 2.3 shows the values of $sc_{\max}$ for acquiring the corresponding solution by RCCPP. For topologies such as Oxford, AT&T, Hibernia, NIIF, and CESNET, both schemes achieve a zero load imbalance with the average controller utilization of 100%.

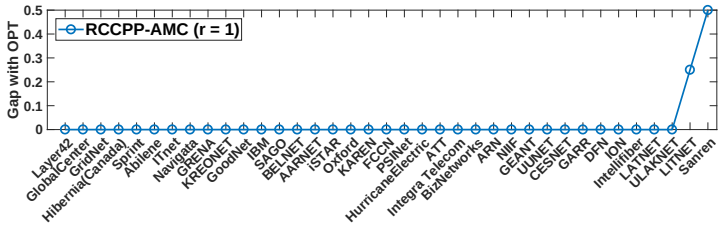
For few topologies, the load imbalance of RCCPP is more than that of CNCP, which mainly results from the trade-off between satisfying the constraint related to $sc_{\max}$ for RCCPP and the load on the controllers. As an example, we consider the Abilene topology with 11 nodes. The set of controllers in a solution obtained by RCCPP is $\{6, 7, 8\}$ with the load imbalance of 1400 $k$req/s whereas it is $\{6, 7, 10\}$ with the load imbalance of 600 $k$req/s for CNCP. The value of $sc_{\max}$ for RCCPP was set to $0.53D_G$, which corresponds to the maximum allowed switch-controller latency of 12.97 ms. Regarding RCCPP, the assignments of the first and second controllers to each switch must satisfy the aforementioned latency, and thus, the maximum switch-controller latency in its solution is 12.86 ms (latency between the switch at node 3 and one of its assigned controllers at node 7). However, the maximum switch-controller latency in the solution obtained by CNCP is 14.71 ms, which results from the assignment of the controller at node 7 to the switch at node 5 as its secondary controller (not a valid assignment in RCCPP). The primary controller of the switch at node 5 is placed at node 6, which is the nearest to node 7 (4.52 ms). This is due to the fact that the objective of CNCP is to minimize the maximum (for all switches) of the sum of the latency from the switch to its first reference controller (which is the nearest controller to the switch with enough capacity) and the latency from the first reference controller to the second reference controller (which is chosen as the closest node to the first reference controller with enough capacity). Thus, using CNCP, the assignment of the controllers to the switches does not necessarily lead to having the lowest possible switch-controller latency which has a higher priority compared with the load imbalance for delay-sensitive applications. Finally, we conclude that RCCPP is more flexible for achieving a good trade-off between the distribution of the load among the controllers and the maximum switch-controller latency. As we

mentioned earlier, in contrast to CNCP, RCCPP is independent from the master controller selection process, simple, and easily extendable, as well as being solvable by a near-optimal and polynomial-time algorithm.

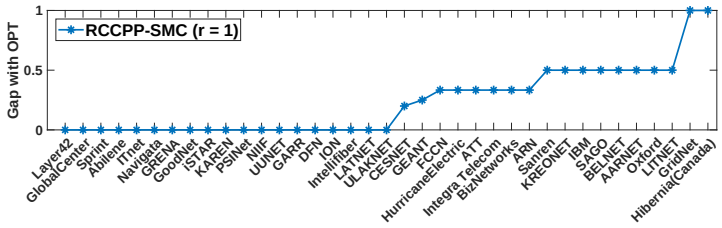
### 2.5.3 Solution Quality of Our Proposed Algorithms

Figure 2.3 shows the (sorted) gap between the results obtained by our proposed algorithms and the optimal solution (OPT) considering a homogeneous traffic load for the switches, $u_c = 2000$ $k$req/s, $cc_{\max} = 0.8D_G$, and $sc_{\max} = 0.6D_G$. We sorted the topologies based on this gap in an increasing order (in each subfigure, the order is different accordingly). The presented results in this figure demonstrate the quality of the solutions acquired by both of the proposed algorithms due to their small gap with OPT (at most 1.5×OPT for RCCPP-AMC and 2×OPT for RCCPP-SMC). For instance, considering the Oxford topology in Figure 2.3b, the gap shown on the y-axis is 0.5, which corresponds to 1.5×OPT. In particular, the value on the y-axis is calculated by subtracting the optimal value from the acquired number of controllers with RCCPP-SMC, divided by the optimal value.

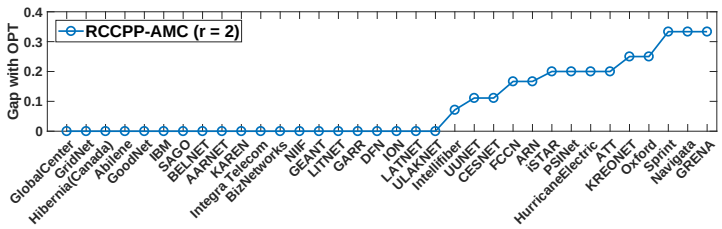
RCCPP-AMC achieves an optimal solution for around 60% of the topologies with regard to resilience against single controller node failures and dual-controller node failures, respectively (Figure 2.3c and Figure 2.3e). However, RCCPP-SMC obtains an optimal solution for around 30% of the topologies for the same failure cases (Figure 2.3d and Figure 2.3f). Obviously, when we have no resilience (Figure 2.3a and Figure 2.3b), both algorithms lead to solutions of higher quality. Note that for very large topologies such as TATA and Cogent, the Gurobi optimizer was unable to obtain the optimal results in a reasonable amount of time. Therefore, we used the equality of the objective value obtained by either of the proposed algorithms and our calculated lower bound (in (2.16)) as an indicator of acquiring an optimal solution. Table 2.4 shows the number of required controllers and the execution time for such topologies and we can see the trade-off between the quality of solutions and the execution time by comparing the obtained values for both algorithms.



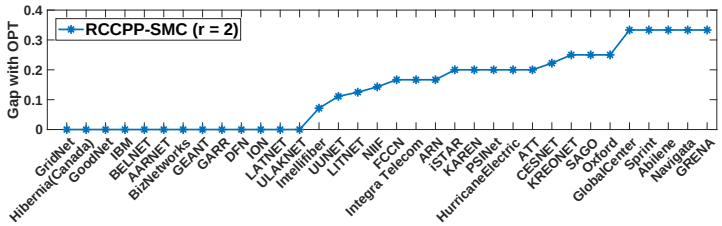

(a)



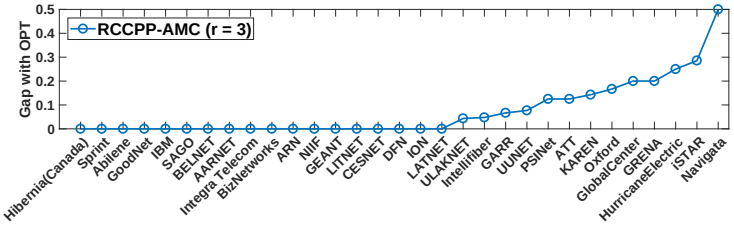

(b)



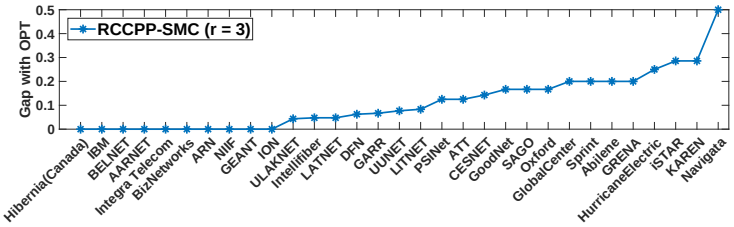

(c)




(d)




(e)




(f)

Figure 2.3: Gap between the proposed algorithms and the optimal solution.

Table 2.4: The value of the objective function and execution time of the proposed algorithms for large topologies

| Topology | Size | Lower Bound | Algorithm | Number of Controllers | Execution Time (s) |
|---|---|---|---|---|---|
| TATA | 143 | 29 | RCCPP-AMC | 29 | 491.07 |
| | | | RCCPP-SMC | 30 | 196.26 |
| Colt Telecom | 146 | 30 | RCCPP-AMC | 30 | 247.39 |
| | | | RCCPP-SMC | 30 | 194.85 |
| Cogent | 180 | 36 | RCCPP-AMC | 36 | 779.46 |
| | | | RCCPP-SMC | 37 | 513.99 |

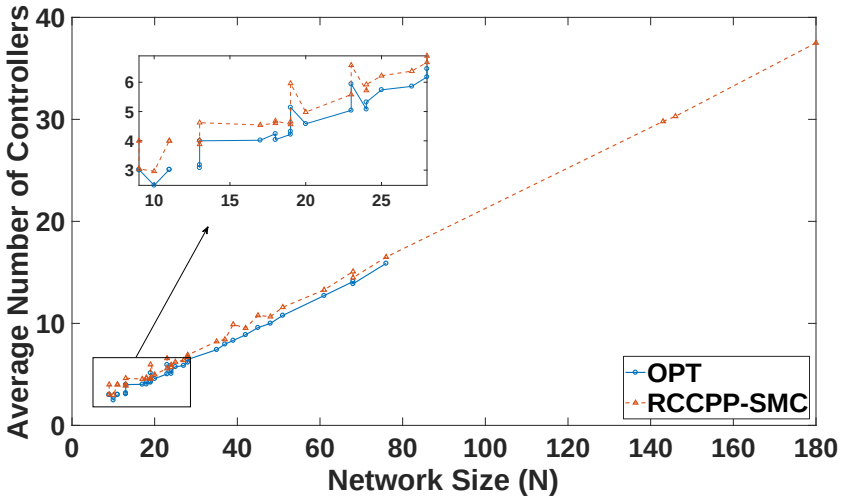



Figure 2.4: Increase in the average number of the controllers with regard to network size for heterogeneous switch traffic loads, $r = 2$, $u_c = 2000$ $k$req/s, $cc_{\max} = 0.8D_G$, and $sc_{\max} = 0.6D_G$.

### 2.5.4 Sensitivity Analysis

#### The impact of topology

We analyze the impact of topology on the number of controllers by considering three aspects, including the size of the topology, its shape and the shortest path lengths.

***The size of the topology***: As shown in Figure 2.4, the average number of controllers increases linearly with regard to the network size ($N$) for both the optimal solution and the solution provided by *RCCP-SMC* for heterogeneous switch traffic loads, $r = 2$, $u_c = 2000$ $k$req/s, $cc_{\max} = 0.8D_G$, and $sc_{\max} = 0.6D_G$ (a similar trend applies to other parameter settings). The average number of controllers for all of the analyzed topologies ranges from around 30% (for $u_c = 2000$ $k$req/s) to 15% (for $u_c = 10,000$ $k$req/s) of their size. It should be noted that for topologies with a very large network size, such as TATA, Colt Telecom, and Cogent, only the results from

RCCPP-SMC are shown since the Gurobi optimizer was unable to obtain the optimal results in a reasonable amount of time (the blue line is discontinued for $N > 80$ in Figure 2.4).

***The shape of the topology***: By examining the results for the topologies of the same size, we found that hub-and-spoke topologies (star-like) usually require more controllers or even the problem tends to become infeasible more easily. Examples of the former case are KREONET with $N = 13$ (the same size as Navigata and GRENA, which are linear topologies) and ARN with $N = 28$ (the same size as BizNetworks which is a linear topology). ITnet with 11 nodes (the same size as Sprint and Abilene which are mesh-like topologies) exemplifies the latter case (i.e., the problem is infeasible) which causes the graph to be disconnected at this point. The reason is mainly due to the higher number of spokes (nodes with degree one). In particular, since more than one controller is assigned to a switch and all the capacity and latency constraints must be satisfied, it is likely to place a controller at a spoke (e.g., the controllers of KREONET are placed at nodes $\{0, 1, 2, 9\}$ out of which 0, 1, and 9 are spokes). If such spokes are far away from the nodes with the highest degrees (such as node 1 in KREONET), they serve a limited number of switches, and thus they increase the number of required controllers. Moreover, most of the hub-and-spoke-like topologies have a low diameter (less than 10 ms) compared with mesh and linear topologies. Hence, they satisfy lower latency bounds but at the expense of having more controllers. It should be noted that although some topologies have large diameters, there is not much room to decrease the values of $cc_{\max}$ and $sc_{\max}$. For example, HurricaneElectric (a linear topology) has a diameter of 147.75 ms; however, no feasible solution is found for $cc_{\max} = 0.6D_G$ (88.64 ms) and $sc_{\max} = 0.4D_G$ (59.09 ms) (for all of the considered controller capacity values), which necessitates the use of graph augmentation to satisfy lower latency bounds with the same parameter settings for traffic loads of switches and capacity of controllers.

***Shortest path lengths***: From the experiments carried out on various topologies with regard to different values of $cc_{max}$ and $sc_{\max}$, we can see that the shortest path lengths between the pairs of nodes have a great impact on the value of the objective function in contrast to the graph density and path redundancies. That is, if the topology has a lot of path redundancies and high density, we cannot conclude that it requires fewer controllers compared with its peers (i.e., topologies of the same size). For instance, GlobalCenter is a complete graph with the highest possible density and highest average node degree for a network size of 9. However, it needs almost the

same number of controllers (for almost all the values of $cc_{max}$ and $sc_{\max}$) as Gridnet, a topology with the same network size but less path redundancy. Another example is Integra Telecom with 27 nodes, which almost has the same size and average node degree as BizNetwoks, but with a higher density.

### The impact of $sc_{\max}$ and $cc_{\max}$

Changing the values of $cc_{\max}$ and $sc_{\max}$ helps the network operators with insights into the impact of such propagation latency bounds on the number of controllers, their respective locations, and the average controller utilization. More importantly, it sheds light on the situation when the problem becomes infeasible as well as the minimum possible propagation latency requirement that can be met by a specific topology without using any graph augmentation techniques or open search [39] (i.e., the controllers can be placed anywhere in the geographical area rather than being co-located with the switches). Also, it is possible to set both $cc_{\max}$ and $sc_{\max}$ with values of interest rather than using $D_G$. For instance, if 50 ms round-trip propagation latency between switches and each of their assigned controllers is needed, then the value of $sc_{\max}$ should be set to 25 ms.

***The average number of controllers***: The results in Figure 2.5 indicate the dominant impact of $sc_{\max}$ compared with $cc_{\max}$ on the average number of controllers for the Sprint topology as a representative of a group of topologies. In particular, for each topology, we kept the value of $cc_{\max}$ fixed and changed the value of $sc_{\max}$. In all of these cases, $\boldsymbol{G_p}$ remains the same. Thus, its density and the number of maximal cliques (in case of using RCCPP-AMC) are unchanged. Lower values of $sc_{\max}$ increase the average number of controllers or even result in infeasibility of the problem. This is due to the fact that limiting the switch-controller latency would lead to fewer potential controllers (to be assigned to a switch), and subsequently it would diminish the possibility of assigning an existing controller in a partial solution to a new switch. However, for some of the topologies such as SAGO, BizNetworks, UUNET and DFN, if we change the value of $cc_{\max}$, the average number of needed controllers remains unchanged regardless of the value of $sc_{\max}$. One reason is that while decreasing the value of $sc_{\max}$, the total number of cliques of $\boldsymbol{G_p}$ (including 2-cliques and 3-cliques) do not vary much for such topologies. These topologies have denser $\boldsymbol{G_p}$ graphs and subsequently have more cliques and potential controller-switch assignments. It should be noted that when $cc_{\max} = sc_{\max} = D_G$, there is no latency requirement for

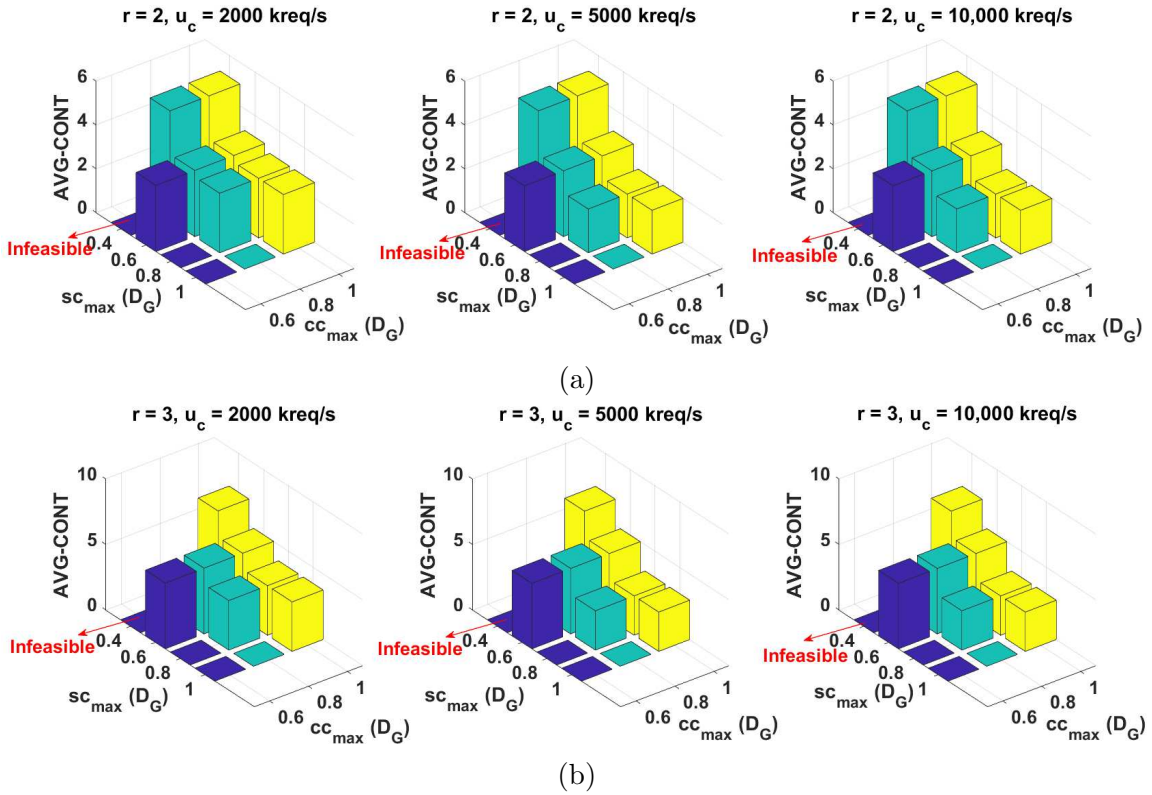
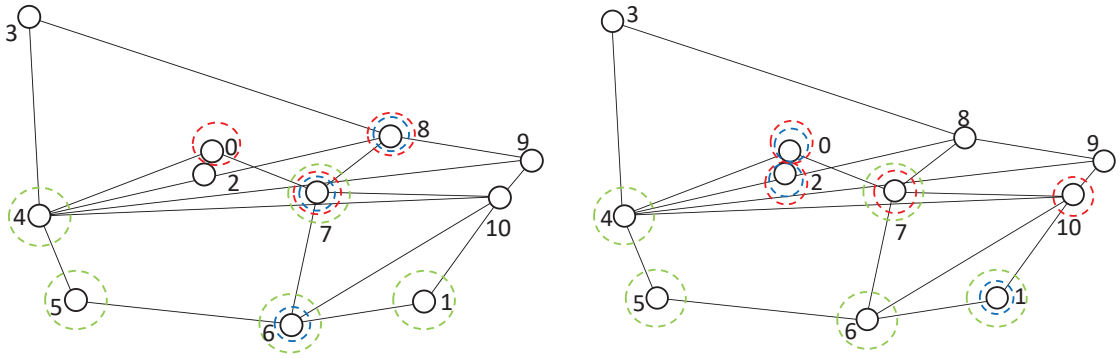



Figure 2.5: The impact of $cc_{\max}$, $sc_{\max}$, and $u_c$ on the average number of required controllers in OPT for the Sprint topology.

the switch-controller latency and inter-controller latency. Furthermore, $cc_{\max} = D_G$ implies no latency requirement for the inter-controller latency (constraint (2.6) is relaxed), and hence $\boldsymbol{G_p} = \boldsymbol{G_o}$ is satisfied since $\boldsymbol{G_o}$ is not pruned. This results in having $\boldsymbol{G_p}$ as a complete graph and finding all maximal cliques is polynomially bounded, and thus RCCPP-AMC obtains a solution in polynomial time. If the original graph $\boldsymbol{G}$ is a complete graph (e.g., GlobalCenter) and $cc_{\max} = D_G$, it is possible that $\boldsymbol{G_p} = \boldsymbol{G_o} = \boldsymbol{G}$. Note that the value of $sc_{\max}$ is upper bounded by $cc_{\max}$, and hence no value is shown for the number of controllers in such cases in Figure 2.5.

***Controller locations***: Figure 2.6 illustrates the controller locations for the Sprint topology as an example of the set of experiments carried out for $r = 2$, homogeneous traffic loads for the switches and $u_c = 2000$ $k$req/s. Since we assume that the controllers have the same capacity, their number is only affected by the traffic loads of switches. On the other hand, both the number and the location of controllers can be affected by the latency bounds. As shown in Figure 2.6a, 3 controllers (in the optimal solution for $cc_{\max} = sc_{\max} = 0.8D_G$) are placed at nodes 6, 7, and 8 (indi-




(a) Sprint (OPT) (b) Sprint (RCCPP-SMC)

Figure 2.6: Controller locations for the Sprint topology. Blue, red and green dashed circles indicate the controller locations for $[cc_{\max} = sc_{\max} = 0.8D_G]$, $[cc_{\max} = 0.8D_G, sc_{\max} = 0.6D_G]$, and $[cc_{\max} = 0.8D_G, sc_{\max} = 0.4D_G]$, respectively.

cated by blue circles). These controllers satisfy the total traffic loads of 11 switches (with $r = 2$, the total load is 4400 $k$req/s). Since the capacity of each controller is 2000 $k$req/s, at least 3 controllers are required. In this case the values of the latency bounds do not affect the number of controllers but only their placements. For instance, the controller located at node 7 (Kansas City), serves the switches at node 4 (Stockton) and node 9 (New York), and the propagation latency between node 7 and the aforementioned switches (11.91 ms and 9.18 ms, respectively) is less than 19.28 ms ($<= 0.8D_G$). Another set of controllers ($\{0, 1, 2\}$), which is also optimal, is acquired by RCCPP-SMC and depicted in Figure 2.6b (denoted by blue circles). However, as an example, if we choose nodes 4 and 10, these two nodes cannot be a member of the set of controllers at the same time or one serves as the controller for the other one since the latency between them (i.e., 19.38 ms) does not satisfy the latency bounds. Decreasing the value of $sc_{\max}$ to $0.6D_G$ does not change the number of controllers (but it changes their placement as shown in Figure 2.6a with red circles) for the optimal solution while RCCPP-SMC provides a solution with 4 controllers (red circles in Figure 2.6b). The green circles in Figure 2.6a and Figure 2.6b show the controller locations when the value of $sc_{\max}$ is reduced to $0.4D_G$, which subsequently affects the number of controllers. Actually, Sprint is one of the few topologies for which reducing $sc_{\max}$ to a small portion of $D_G$ results in costly solutions in terms of the number of the required controllers (almost 50% of the network size). This case usually happens for small topologies.

***Controller utilization***: Table 2.5 shows the average controller utilization for

Table 2.5: Average controller utilization (Sprint)

| $(cc_{\max}, sc_{\max})$ | Resilience Level | $u_c = 2000$ *k*req/s | $u_c = 5000$ *k*req/s | $u_c = 10,000$ *k*req/s |
|---|---|---|---|---|
| $(D_G, D_G)$ | r=2 | 80.6% | 43.5% | 21.74% |
| | r=3 | 86.35% | 43.5% | 21.74% |
| $(D_G, 0.8D_G)$ | r=2 | 80.6% | 43.5% | 21.74% |
| | r=3 | 86.35% | 43.5% | 21.74% |
| $(D_G, 0.6D_G)$ | r=2 | 71.9% | 29% | 14.5% |
| | r=3 | 64.9% | 26.09% | 13.04% |
| $(D_G, 0.4D_G)$ | r=2 | 43.5% | 17.4% | 8.7% |
| | r=3 | 46.41% | 18.64% | 9.32% |
| $(0.8D_G, 0.8D_G)$ | r=2 | 80.6% | 43.5% | 21.74% |
| | r=3 | 86.35% | 43.5% | 21.74% |
| $(0.8D_G, 0.6D_G)$ | r=2 | 71.9% | 29% | 14.5% |
| | r=3 | 64.9% | 26.09% | 13.04% |
| $(0.8D_G, 0.4D_G)$ | r=2 | 43.5% | 17.4% | 8.7% |
| | r=3 | - | - | - |
| $(0.6D_G, 0.6D_G)$ | r=2 | 71.9% | 29% | 14.5% |
| | r=3 | 64.9% | 26.09% | 13.04% |
| $(0.6D_G, 0.4D_G)$ | r=2 | - | - | - |
| | r=3 | - | - | - |

the Sprint topology in the optimal solution with regard to different values of $cc_{\max}$ and $sc_{\max}$ and controller capacities (for $r = 2$). Note that the dashes in the table indicate the infeasibility of the problem. The 2-tuples in the first column show the values of $cc_{\max}$ and $sc_{\max}$ as a ratio of $D_G$. If the values of $cc_{\max}$ and $sc_{\max}$ are reduced, both the number of controllers and their assignments to the switches change, and therefore they affect the average controller utilization. Similarly, for a fixed value of $cc_{\max}$, lowering the value of $sc_{\max}$ reduces the average controller utilization. Other topologies such as DFN, BizNetworks, ULAKNET, and TATA have the same average controller utilization regardless of the values of $cc_{\max}$ and $sc_{\max}$, which is consistent with having the same average number of controllers. For such topologies, the average controller utilization is around $80-90\%$ for resilience against single and dual-controller node failures.

***The infeasibility of the problem***: For most of the topologies, infeasibility is mainly caused by lower values of $sc_{\max}$ (no controller-switch assignment could be found to satisfy all of the constraints in the problem). Nevertheless, for a few topologies such as Integra Telecom, the problem becomes infeasible when $cc_{\max} = 0.6D_G$ regardless of the value of $sc_{\max}$. This is due to the fact that no feasible solution can be found for any of the maximal cliques of $\boldsymbol{G_p}$. Moreover, as we mentioned in Section 2.4, one reason for the infeasibility of the problem is that $\boldsymbol{G_p}$ becomes disconnected (which is related to the value of $cc_{\max}$). This happens for topologies

such as iSTAR, BizNetworks, Sanren, Gridnet, GEANT (2012), ITnet, ARN, FCCN, LATNET, and ULAKNET, if we set $cc_{\max} = 0.4D_G$. However, as shown in Table 2.2, we assume that the value of $cc_{\max}$ is at most $0.6D_G$ (in line with the existing work such as [30]).

**The impact of controller capacity ($u_c$)**

While increasing $u_c$ leads to a lower number of controllers (more than 50% decrease) for some of the topologies such as ULAKNET, TATA and Cogent, it does not necessarily cause a decreasing trend for the others such as Sanren and Sprint (topologies with a small size). For instance, as illustrated in Figure 2.5a, regardless of the capacity of the controllers, 5 controllers are needed when $cc_{\max} = D_G$ and $sc_{\max} = 0.4D_G$. However, the maximum total traffic loads of all switches is lower than the total capacity provided by only 3 controllers. This mainly results from the reduced value of $sc_{\max}$, i.e., $0.4D_G$ compared with the scenario in which $cc_{\max} = D_G$ and $sc_{\max} = 0.6D_G$. Therefore, the number and set of the controller nodes that satisfy the switch-controller latency are mostly different from each other in the aforementioned two scenarios. The controllers are at nodes {1, 4, 5, 8, 9} in most of the experiments for the former case while the controllers are at nodes {0, 6, 8} for the latter case. A similar trend is observed for the results obtained by RCCPP-SMC with 6 and 4 controllers on average for the former and the latter cases, respectively. Moreover, the capacity of controllers has a reciprocal relationship with the average controller utilization. The reason is that by increasing the controller capacities for the same amount of the total load, the controllers are under-utilized (as shown in Table 2.5). It should be noted that we have used practical values for the capacity of controllers (as in Table 2.2) and the value of capacity is usually much higher than the traffic loads of switches. Therefore, $u_c$ is not a restrictive factor that affects the feasibility of the solutions obtained by the proposed algorithms whereas $cc_{\max}$ and $sc_{\max}$ have a more dominant role in this case as we discussed before.

### 2.5.5 Single Link Failures

As we discussed in Section 2.3.3, our problem formulation can be extended to include resilience against single link failures in addition to controller node failures. Although having link-disjoint paths between a switch and all of its assigned controllers (whether the primary controller or the backup ones) may result in infeasibility for some of the

problem instances, it is needed to assure the tolerance of single link failures. This infeasibility issue can be alleviated by using graph augmentation mechanisms or at least maximizing the number of such link-disjoint paths. It should be noted that the aforementioned constraint can be easily incorporated into both of the proposed algorithms (by choosing the $r$ or $(r+1)$-cliques that contain link-disjoint paths between a switch and its assigned controllers after obtaining $\boldsymbol{G_p}$). Since this is not our key focus in this chapter, here we just report some of our observations when applying this additional constraint to our problem formulation. Mesh topologies tend to be less affected by the infeasibility issue due to their higher density and path redundancy (the number of controllers increases for some values of the latency bounds and few of the previously feasible problem instances become infeasible) whereas hub-and-spoke and linear topologies are affected, especially when the value of $cc_{\max}$ is decreased. For instance, considering the DFN topology (a mesh topology with $N = 51$), applying the link-disjoint path constraint results in the infeasibility of the problem for $r = 2$, $cc_{\max} = 0.8D_G$ and $sc_{\max} = 0.4D_G$ as well as causing no increase in the maximum number of assigned controllers for other scenarios. On the other hand, for the CESNET topology (a hub-and-spoke topology with $N = 45$), the problem becomes infeasible for $r = 2$ and $cc_{\max} < D_G$ as well as the maximum number of assigned controllers is increased by 3 times for other scenarios.

## 2.6 Conclusions

In this chapter, we have proposed two algorithms for solving RCCPP, which provide high-quality and cost-saving solutions. By finding the maximal cliques, we narrow down the search space and the set of controllers in an optimal solution (if any) is always a subset of one of the maximal cliques. The effectiveness of the proposed algorithms was extensively analyzed with regard to various parameter settings for a wide range of real WAN topologies. Such an analysis can assist network operators with helpful insights into the design and management of their SDN-based networks to meet different SLAs. The proposed algorithms can be easily amended to cover node or link failures even with different objective functions (e.g., minimizing the expected control path loss). Another direction is to look into the dynamic RCCPP which changes the controller-switch assignments based on the time-varying traffic loads of switches. We will investigate this in the next chapter.

# Chapter 3

# Resilient Switch Reassignment and Incremental Controller Placement for Software-Defined WANs

## 3.1 Background

As we discussed in Chapter 2, due to the strong reliance of SDN on logically centralized controllers, one of its key challenges is the CPP. In the static CPP, which is more practical to be solved at the stage of initial network design, the switches are assigned to the controllers by considering a set of criteria. Such criteria include the QoS requirements (such as the switch-controller and inter-controller latencies), the capacity of the controllers along with the average, maximum or approximated traffic loads of switches, and resiliency. As we showed in Chapter 2, any change in the aforementioned criteria can result in a different controller placement (different number of controllers, controller locations, and switch-controller assignments).

The adjustment of controller placement due to change in the switch-controller latency requirement or the level of resiliency in SD-WANs can subsequently lead to a change of the load on the existing controllers. A more viable and legitimate reason behind modifying controller placement is varying traffic loads of switches. The SD-WANs traffic matrices and flow generation rates are time-varying. In real network scenarios, the load on a controller depends on the number of its connected switches and the frequency at which these switches encounter new flows [39]. In particular, if the traffic load (mainly the flow setup requests) of one or more switches connected to

a controller increases, it may cause the controller to be overloaded and subsequently increases its response time. This is critical especially when the controller operates in a reactive mode or a hybrid mode (which is a combination of reactive and proactive flow setup) with regard to flow installation in switches using the OpenFlow protocol. In the reactive mode, the controller adds, updates, and deletes flow entries in flow tables reactively (in response to packets) [20]. Similarly, if the traffic load decreases, it may lead to the under-utilization of the controller. Moreover, the evolution of Network Function Virtualization (NFV), as a complementary technology to SDN, necessitates the need for a distributed cloud that enables the insertion, removal, and migration of Virtual Machines (VMs) across different geographical locations [78]. Various services running in data centers produce different traffic patterns and a large number of flows with different SLAs, which cause frequent VM migration and subsequently a considerable amount of migration traffic [79]. As a result, an SD-WAN that interconnects the data centers must be able to handle such dynamically varying traffic patterns while maintaining their resiliency requirements (if any). To achieve this, controller placement should be changed accordingly in a periodic or event-driven manner.

To lower the cost of change in the controller placement, the service providers prefer to reuse the existing deployed controllers as much as possible. Therefore, to properly deal with the variation of the traffic loads of switches or change in the QoS requirements, which subsequently affects the load balancing among the controllers, two types of solutions are introduced in this chapter.

The first possible approach is *switch reassignment*, which involves changing the current switch-controller assignments. The reassignment of switches to the current controllers may lead to an increase or decrease in the number of active controllers (active controllers are the ones that have at least one switch connected to them) or just a change in the assignment of the existing active controllers to the switches. However, the reassignment of switches incurs additional operating and maintenance costs. Therefore, frequent switch reassignment is not desirable for the service providers. Using a central entity, *switch reassignment* (as a form of network re-optimization) should be performed periodically or triggered when the load on a controller is lower than or exceeds a predefined threshold (to handle the possible controller under/overload) or if there is a change in the switch-controller latency requirement.

If the aforementioned approach is not sufficient, the second approach is *incremental controller placement*. Through this, one or more controllers are added to the existing controllers to satisfy the change in the switch traffic patterns (that can assist

in the alleviation of possible controller overload) and/or new QoS requirements. In this case, we should decide the number of such additional controllers and their locations in the SD-WAN topology. It should be noted that the rate of change in the traffic patterns in SD-WANs is not as fine-grained and dynamic as in a data center. Therefore, periodic/event-driven solutions suffice for SD-WANs rather than online algorithms such as [80, 81] that suit the handling of dynamics within a data center.

The contributions of this chapter are threefold. First, we have formulated the *resilient switch reassignment* (without/with budget constraints) and *incremental controller placement* problems that stem from the dynamics of the system (e.g., the variations of the traffic loads of switches) for a given controller placement. Second, we have provided algorithms with guaranteed bounds by modeling the aforementioned problems (via grouping and/or decomposing the problem) as a variant of the classical problem of *scheduling on parallel machines* which incorporates *machine eligibility constraints* (i.e., each job can only be processed by a subset of machines), *arbitrary processing sets* (i.e., any set relationship between the processing sets of each pair of the jobs), and *resiliency*. To the best of our knowledge, we are the first one to address and solve such a problem and the provided solutions are not restricted to the context of controller placement. Third, we have conducted an extensive analysis of the results on real WAN topologies to give detailed insights about the problems.

## 3.2 Related Work

In this section we first give an overview on the topic of switch reassignment in SDN-enabled networks. Then, we review the most recent work on scheduling with machine eligibility and processing set restrictions, which is the underlying mathematical problem that assists us in solving the switch reassignment problem.

### 3.2.1 Switch Reassignment

There are two main lines of work on dynamic switch reassignment in SDN-based networks, namely intra-data center solutions and inter-data center (in SD-WANs) solutions. In the former, the controller CPU processing time is the key contributor to the flow setup latency and also controller traffic overhead is important (since the controller-switch communication is frequent and occupies scarce bandwidth resources) [80]. Moreover, the traffic pattern of switches is more fine-grained with con-

siderably high variations. For this reason, online algorithms [80,81] are well-suited to this category. In contrast, in SD-WANs, the propagation latency (in order of millisecond) is the main contributor to the flow setup latency. In the following, we review the existing work on the dynamic switch reassignment and controller placement in SD-WANs, which is the focus of this chapter.

Huque et al. [39] proposed an algorithm (called LiDy+) for large-scale dynamic controller placement. In particular, using the smallest enclosing disk algorithm, LiDy+ determines the controller placements with regard to the switch-controller latency bound. Then, it runs a dynamic flow management module at each controller location to handle the load balancing among multiple controllers locally. Therefore, the term *dynamic* in this work means changing the number of controllers at a specific location with regard to the change in the traffic loads of switches, which does not reflect the dynamic switch-controller assignments. Also, no inter-controller latency was considered. In [35], the authors proposed a Dynamic Controller Provisioning Problem (DCPP) which changes the number and locations of the controllers with regard to varying network conditions to minimize the flow setup time and communication overhead in a dynamic manner. The reassignment was applied periodically to adjust both the number of controllers and their locations based on two heuristic algorithms (namely, greedy knapsack and simulated annealing). In contrast, our proposed model is aimed at reusing the existing controllers as much as possible by changing the switch-controller assignments. Otherwise, it adds a number of controllers incrementally to adapt to the new requirements. Moreover, we have proposed approximation algorithms to solve the problem.

The proposed controller placement problem in [82] was formulated as a quadratic program with the goal of minimizing the controller-switch latency and maximizing the connectivity between the switches and their associated controller. Starting with a static controller placement obtained by solving the aforementioned optimization problem, the authors proposed another quadratic program along with a heuristic algorithm for switch migration to adjust the load on the controllers. They assessed the performance of their scheme with regard to the number of overloaded controllers and the number of migrated switches on small and medium-size real topologies. Hegde et al. [83] considered an edge-core SDN architecture with source routing. A dynamic controller load balancing approach was utilized for the edge network which involved periodically checking the load on a controller. In case the load on a controller exceeds a threshold, the switch migration was carried out heuristically by only reassigning a

chosen switch (based on its degree in the graph and its latency to the controller) to the least loaded controller without taking the reassignment cost into account. It should be noted that in all of the the aforementioned work in this subsection, controller capacity and resiliency were not considered at the same time as well as no bounded solution was proposed.

### 3.2.2 Scheduling with Processing Set Restrictions

For the most basic form of scheduling on identical parallel machines, i.e., without processing set restrictions (each job can be scheduled on any machine), there are a number of approximation algorithms such as a 2-approximation greedy list scheduling algorithm and a $\frac{4}{3}$-approximation algorithm [84]. Leung et al. [85, 86] provided a survey of the most recent literature on the scheduling with processing set restrictions. There are special forms of processing set restrictions such as inclusive processing sets, nested processing sets, and interval processing sets. However, the processing set of a job can be an arbitrary subset of the set of all machines in the most general case of processing set restrictions. In this chapter, we are interested in scheduling on parallel machines with arbitrary processing sets since it fits our problem (in the switch reassignment, the potential controller sets for the switches can have any set relationship with each other).

In the most recent work [87, 88], the authors analyzed the jump neighborhood (a jump move involves moving a job from its currently assigned machine to another machine in its processing set) for the purpose of scheduling on identical parallel machines with arbitrary processing sets by proving the worst-case performance bounds. In particular, they showed that a jump-optimal assignment for such a problem has the makespan of at most $\frac{1}{2} + \sqrt{m - \frac{3}{4}}$ times the optimal makespan ($m$ is the number of machines). However, the aforementioned gap with the optimal solution rises when the number of machines increases. Therefore, we also reviewed other work on scheduling for which there exist constant factor approximation algorithms. In particular, [89] provides a 2-approximation algorithm for scheduling jobs on unrelated parallel machines. Reviewing the usage of the aforementioned algorithm in some of the recent work, e.g., [90] (the access point association was modeled as a scheduling problem), inspired us to conclude that such an algorithm (with slight modifications) is applicable for arbitrary processing sets. In this chapter, we also incorporated the resiliency (i.e., considering more than one machine for a job to be scheduled on) into

the problem by using a grouping strategy in Section 3.3.2.

## 3.3 System Model and Problem Formulation

### 3.3.1 System Model

The topology of an SD-WAN is represented by a connected graph $\boldsymbol{G}$=($\boldsymbol{V}$, $\boldsymbol{E}$), where $\boldsymbol{V} = \boldsymbol{S}$, and $\boldsymbol{S}$ is the set of OpenFlow-enabled switches, while $\boldsymbol{E}$ denotes the set of weighted links. The weights of the links are the propagation latencies (shortest path lengths) between the nodes based on their geographical locations. Given a controller placement for an SD-WAN topology, one can extract the current set of switch-controller assignments $\boldsymbol{A}$ and the current existing controllers (denoted by set $\boldsymbol{C}$). Table 3.1 shows the notation used in the problem formulation and proposed solutions.

We assume that the controllers share the same location with the switches, i.e., some of the nodes host both a switch and a controller. Furthermore, we assume all the geographically distributed controllers are identical with a uniform maximum total capacity. A *controller* is a logical entity, which represents a set of controller software instances running on one or more physical servers and their total capacity determines the controller capacity at a particular geographical location. The potential reasons behind *switch reassignment* and *incremental controller placement*, i.e., the change in the switch-controller latency requirement and the variation of the traffic loads of switches, can subsequently affect the load balancing among the controllers. If a controller becomes overloaded, the processing latency increases and becomes a non-negligible part of the total switch-controller latency. Moreover, an overloaded controller has a higher possibility of failure. Therefore, our objective is to minimize the maximum load on the controllers.

Figure 3.1 depicts our proposed controller placement framework, which contains four modules. The *Controller Load Monitoring Module* monitors the load on the controllers and triggers the *Switch Reassignment Module* if the load on at least one controller exceeds or falls behind the pre-defined threshold for the allowed maximum or minimum load. It also periodically decides whether to perform the switch reassignment or not based on the historical data and other collected statistics collected for a certain network topology. New changes in the system are specified in the *New Requirements Module*, which subsequently calls the *Switch Reassignment Module*. When

Table 3.1: Main notation used in the problem formulation and proposed solutions

| Symbol | Definition |
|---|---|
| $\boldsymbol{G}$ | Topology of an SD-WAN |
| $\boldsymbol{S}$ | Set of OpenFlow-enabled switches |
| $\boldsymbol{S_k}$ | Set of switches in class $k$ |
| $\boldsymbol{C}$ | Set of controllers in the current controller placement |
| $\boldsymbol{C_i}$ | Set of the potential controllers for switch $i$ |
| $\boldsymbol{A}$ | Set of all assignments in a controller placement |
| $\boldsymbol{L}$ | Set of the traffic loads of switches |
| $\boldsymbol{D}$ | Set of dissociated switches |
| $\boldsymbol{D_k}$ | Set of dissociated switches in class k |
| $\boldsymbol{B}$ | Bipartite graph that shows both complete and fractional assignments |
| $\boldsymbol{H}$ | Bipartite graph that shows fractional assignments |
| $\boldsymbol{M}$ | A matching in $\boldsymbol{H}$ |
| $\boldsymbol{Q}$ | Set of available controllers, i.e., a superset of $\boldsymbol{C}$ |
| $c_i$ | The controller that is currently associated with switch $i$ |
| $D_G$ | Diameter of graph G |
| $sc_{\max}$ | Latency bound between a switch and its assigned controllers |
| $cc_{\max}$ | Inter-controller latency bound |
| $l_i$ | Traffic load of switch $i$ |
| $l_{ij}$ | Load of switch $i$ on controller $j$ |
| $r$ | Number of controllers to serve a given switch (resilience parameter) |
| $d_{ij}$ | Minimum propagation latency between node $i$ and node $j$ |
| $y_{ij}$ | 1 if switch $i$ is dissociated from its current controller and associated to controller $j$, and 0 otherwise |
| $x_{ij}$ | 1 if switch $i$ is currently connected to the controller at node $j$, and 0 otherwise |
| $\tilde{x}_{ij}$ | 1 if switch $i$ is assigned to controller $j$ after performing the reassignment, and 0 otherwise |
| $z_i$ | 1 if switch $i$ is dissociated from its current controller, and 0 otherwise |
| $w_{ij}$ | 1 if switch $i \in \boldsymbol{D}$ is reassigned to controller $j$, and 0 otherwise |
| $\delta$ | Pre-determined budget for the number of reassignments |
| $\gamma$ | Maximum pre-determined increment parameter in ICPP |
| $\phi$ | Maximum load on the controllers |
| $\phi_j^-$ | Load on controller $j$ after switch dissociation |
| $\phi_j^+$ | Load on controller $j$ after switch reassignment |
| $\lambda$ | The value of the objective function for a feasibility problem |

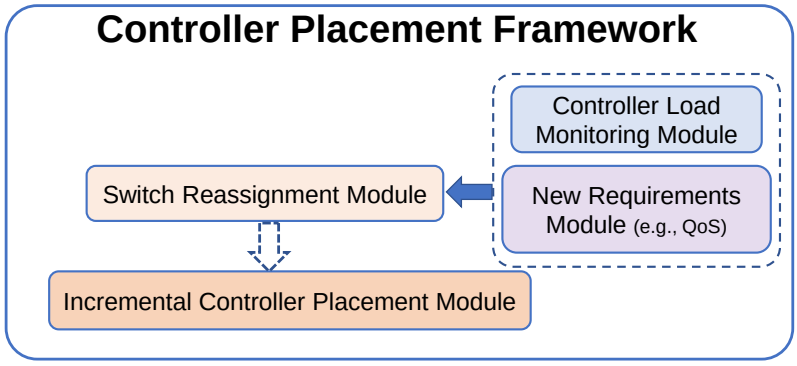



Figure 3.1: Our proposed controller placement framework.

the *Switch Reassignment Module* is triggered, all/some switches are dissociated from their current controllers and reassigned to other controllers while satisfying the constraints.

We define two types of *switch reassignment*. The first type of reassignment is called *unbudgeted resilient switch reassignment*, which does not impose any limitation on the number of reassigned switches. Therefore, in the resultant controller placement, all or some of the switches have been reassigned. The second form of *switch reassignment* is called *budgeted resilient switch reassignment* and it restricts the number of *switch reassignments*. In both types of the *switch reassignment* the number of controllers is fixed, i.e., the service provider prefers to utilize the existing set of controllers as much as possible when handling the dynamic traffic loads of switches or satisfying new QoS requirements. If *switch reassignment* is inadequate, the *Incremental Controller Placement Module* is activated which adds a number of new controllers to the set of existing controllers to satisfy the controller placement constraints and balance the load among the controllers. More details are given in the following sections.

It is worth mentioning that very frequent reassignment of switches can cause instability in the system. Also, the process of switch reassignment can result in service disruption. While the former can be addressed by tuning the system parameters, the latter has already been addressed in the existing work such as [15, 91, 92]. In this chapter our key focus is on how to adapt the existing controller placement mainly through switch reassignment with regard to the changes in the system in a cost-effective manner and solve the associated optimization problems using approximation algorithms. However, we provide some clarifications in the following.

While very frequent reassignments can cause instability in the system, performing the reassignments with very low frequency cannot capture the change in the system and may lead to overload of the system. Also, as we mentioned in Section 3.2.1, the rate of change in the traffic patterns in SD-WANs is not as fine-grained and dynamic as in a data center, which makes the instability less likely to happen in the system. The frequency of the switch reassignment is determined by the monitoring module during the lifespan of the network with regard to the state of the system as well as the network topology characteristics (e.g., large network topologies usually require less frequent switch reassignments since solving the switch reassignment problem takes longer time). Moreover, regarding the events that can cause switch reassignment or incremental controller placement, a time interval can be set to prevent very frequent reassignments and changes in the controller placement.

Furthermore, the method to achieve switch reassignment necessitates coordination between the source and target controllers and the switch [13]. OpenFlow does not provide native support for disruption-free switch migration. Therefore, as we mentioned earlier, some existing work such as [15, 91, 92] addressed this issue by proposing disruption-free switch migration protocols. For instance, [15] provides a 4-phase migration protocol, which is OpenFlow standard compliant, to keep the possible disruption to ongoing flows minimum. The main underlying idea is to generate a single trigger to indicate the exact moment of handoff between the source controller (the initial master controller) and target controller. This process includes changing the role of the target controller to "equal mode" (in OpenFlow), the insertion and removal of a dummy flow in the switch by the source controller, flushing the pending requests in the switch using the OpenFlow's barrier message, and finally changing the role of the target controller to "master mode". Similar to [13], we assume that the reassignment module adopts a switch migration protocol for disruption-free switch migration between controllers during reassignment.

### 3.3.2 Unbudgeted Resilient Switch Reassignment Problem (URSRP)

In URSRP, we should determine the new assignments of switches to the controllers such that the switch-controller latency and resiliency constraints are satisfied while the maximum load on a controller is minimized. We define a binary variable, $\tilde{x}_{ij}$, which is set to 1 if switch $i$ is assigned to controller $j$ after performing the reassignment and 0 otherwise. The load on a controller is defined as the total traffic loads of switches (i.e., mainly the flow setup requests) served by it in a switch-controller assignment. Such a load for a controller $j$ is denoted by $\phi_j$, and it is given by the relation in (3.1),

$$\phi_j = \sum_{i \in \boldsymbol{S}} l_i \tilde{x}_{ij}, \tag{3.1}$$

where $l_i$ is the traffic load of switch $i$. Thus, our objective is to minimize $\phi = \max_j \phi_j$. URSRP is formulated as an IP (Integer Programming) problem, called $P_1$, as follows.

$$P_1: \quad \min \phi \tag{3.2}$$

subject to

$$\sum_{j \in \boldsymbol{C_i}} \tilde{x}_{ij} = r, \quad \forall i \in \boldsymbol{S} \tag{3.3}$$

$$\tilde{x}_{ij} \in \{0, 1\}, \quad \forall i \in \boldsymbol{S},\ \forall j \in \boldsymbol{C}. \tag{3.4}$$

The constraint in (3.3) ensures that each switch is connected to exactly $r$ controllers. If $r > 1$, a switch is assigned to multiple controllers and thus resiliency is embedded in the problem formulation. Moreover, $\boldsymbol{C_i}$ denotes the set of potential controllers for switch $i$, i.e., the propagation latency between switch $i$ and any of the controllers in $\boldsymbol{C_i}$ is less than or equal to the switch-controller latency threshold. The integrality constraints of the problem are provided in (3.4).

To assist in the formulation of the budgeted version of the URSRP in Section 3.3.3 as well as providing a solution by mapping it to a scheduling problem in Section 3.4 while incorporating the resiliency (i.e., $r > 1$), we give an equivalent form for the above formulation in the following. We first create $r - 1$ additional copies of each switch. This is due to the fact that each switch must be assigned to $r$ distinct controllers (which corresponds to having $r$ assignments for each switch). Then, we use a grouping strategy by grouping the switches into $|\boldsymbol{S}|$ classes where each class includes $r$ copies of a switch $i$ (i.e., $\boldsymbol{S}_k = \{s_1^k, s_2^k, ..., s_r^k\}$), and we call the new set of switches $\boldsymbol{S'} = \cup_{k=1}^{|\boldsymbol{S}|} \boldsymbol{S}_k$. The new formulation resulting from this modification is as follows.

$$P_1' : \quad \min \phi \tag{3.5}$$

subject to

$$\sum_{j \in \boldsymbol{C_i}} \tilde{x}_{ij} = 1, \quad \forall i \in \boldsymbol{S'} \tag{3.6}$$

$$\tilde{x}_{i'j} + \tilde{x}_{i''j} \leq 1, \quad \forall i', i'' \in \boldsymbol{S}_k, k \in \{1, 2, ..., |\boldsymbol{S}|\},\ \forall j \in \boldsymbol{C}. \tag{3.7}$$

$$\tilde{x}_{ij} \in \{0, 1\}, \quad \forall i \in \boldsymbol{S'},\ \forall j \in \boldsymbol{C}. \tag{3.8}$$

The constraint in (3.7) ensures that no switches from the same class are assigned to the same controller.

### 3.3.3 Budgeted Resilient Switch Reassignment Problem (BRSRP)

Given $\boldsymbol{S}$, $\boldsymbol{C}$, and $\boldsymbol{A}$, BRSRP determines the reassignment of switches to the controllers such that the switch-controller latency, the resiliency level, and the reassignment limit are met. Here, we define a binary variable $y_{ij}$ in addition to $\tilde{x}_{ij}$, such that $y_{ij}$ indicates whether switch $i$ is dissociated from its current controller and re-associated with a different controller $j$ (i.e., it is set to 1 if $(x_{ij},\ \tilde{x}_{ij}) = (0,\ 1)$ and 0 otherwise). The formulated problem is given by

$$P_2: \quad \min \phi \tag{3.9}$$

subject to (3.6), (3.7), and

$$\sum_{j \in \boldsymbol{C_i}} \sum_{i \in \boldsymbol{S'}} y_{ij} \leq \delta \tag{3.10}$$

$$\tilde{x}_{ij}, y_{ij} \in \{0, 1\}, \quad \forall i \in \boldsymbol{S'},\ \forall j \in \boldsymbol{C}. \tag{3.11}$$

The constraint in (3.10) expresses that the number of reassignments is less than a predetermined budget $\delta$ ($\delta \leq |\boldsymbol{A}|$ where $|\boldsymbol{A}|$ is the number of assignments in the controller placement) while (3.11) provides the integrality constraints. It should be noted that removing the reassignment cost constraint in (3.10) leads to the reduction of BRSRP to URSRP. It is obvious that if $r = 1$, we will have $\boldsymbol{S} = \boldsymbol{S'}$ and there is no need for the constraint in (3.7). To ease the process of finding a solution and being inspired by [90], we divide BRSRP into two sub-problems, namely *switch dissociation* and *switch reassignment*, which are described in the following.

#### Switch Dissociation

In this sub-problem, we indicate which switches are dissociated from their current controllers (the set of such switches is denoted by $\boldsymbol{D}$ and it is a subset of $\boldsymbol{S'}$). We define a binary variable $z_i$ which determines whether switch $i$ is dissociated from

its current controller (denoted by $c_i$). The load on a controller $j$ after the switch dissociation is denoted by $\phi_j^-$, which is given as follows.

$$\phi_j^- = \phi_j - \sum_{i \in \boldsymbol{S'}:\ c_i = j} l_i z_i. \tag{3.12}$$

Since we aim at decreasing the maximum load on a controller as much as possible, the *switch dissociation* problem is given by

$$P_{2-1}: \quad \min \phi^- \tag{3.13}$$

subject to

$$\sum_{i \in \boldsymbol{S'}} z_i \leq \delta \tag{3.14}$$

$$z_i \in \{0, 1\}, \tag{3.15}$$

where $\phi^- = \max_j \phi_j^-$.

**Switch Reassignment**

In this phase, we determine the new controllers to which the dissociated switches should be reassigned. It should be noted that if $r > 1$ for the reassignment of a switch in $\boldsymbol{D}$ to a controller, we must take the constraint in (3.3) into account. That is, the reassignment must be performed such that each switch is connected to $r$ distinct controllers in the resultant assignment. Moreover, after solving the *switch reassignment* sub-problem, the load on a controller $j$ is the sum of $\phi_j^-$ and the total amount of load incurred on this controller resulting from *switch reassignments*. This load is denoted by $\phi_j^+$ and is given by

$$\phi_j^+ = \phi_j^- + \sum_{i \in \boldsymbol{D}} l_i w_{ij}, \tag{3.16}$$

where $w_{ij}$ is a binary variable that is set to 1 if switch $i$ in $\boldsymbol{D}$ is reassigned to controller $j$ and 0 otherwise. The objective of the *switch reassignment* sub-problem is to minimize the maximum load on the controllers, i.e., $\min \max_j \phi_j^+$. It is obvious that $\phi = \max_j \phi_j^+$

and we formulate the *switch reassignment* sub-problem as follows.

$$P_{2-2}: \quad \min \phi \tag{3.17}$$

subject to

$$\sum_{i \in \boldsymbol{D}} l_i w_{ij} \leq \phi - \phi_j^-, \quad \forall j \in \boldsymbol{C} \tag{3.18}$$

$$\sum_{j \in \boldsymbol{C_i}} w_{ij} = 1, \quad \forall i \in \boldsymbol{D} \tag{3.19}$$

$$w_{i'j} + w_{i''j} \leq 1, \ \forall i', i'' \in \boldsymbol{D}_k(|\boldsymbol{D_k}| > 1), k \in \{1, 2, ..., |\boldsymbol{D}|\}, \ \forall j \in \boldsymbol{C} \tag{3.20}$$

$$w_{ij} \in \{0, 1\}, \ \forall i \in \boldsymbol{D}, \ \forall j \in \boldsymbol{C}. \tag{3.21}$$

It is possible that a switch is disconnected from less than $r$ controllers during the dissociation process. Thus, $\boldsymbol{D}_k \subset \boldsymbol{S_k}$ and the constraint in (3.20) only applies to all $\boldsymbol{D}_k$ that include more than one switch.

### 3.3.4 Incremental Controller Placement Problem (ICPP)

The ICPP not only decides the number of the new controller instances and their locations but also indicates the reassignment of switches to the controllers subject to a similar set of constraints to URSRP. The new controllers are chosen from a set $\boldsymbol{Q'}$, which indicates the set of available controllers by excluding the set of existing controllers from a set $\boldsymbol{Q}$. This set is determined at the design stage of the network when the controllers are placed in a static manner with regard to different criteria such as inter-controller latency requirement and the cost of the deployment. In the simplest form, we can assume $\boldsymbol{Q} = \boldsymbol{S}$, i.e., any switch location can serve as a potential location for a controller. This is the case if no inter-controller latency threshold is considered or the inter-controller latency threshold is higher than the diameter of $\boldsymbol{G}$. To solve the initial static controller placement problem (its resultant controller placement is the

input of URSRP, BRSRP, and ICPP), we used the approach in Chapter 2. It results in a controller placement in which the set of controllers ($\boldsymbol{C}$) is a subset of a maximal clique of a pruned graph $\boldsymbol{G_p}$ (based on the inter-controller latency threshold) which is built on $\boldsymbol{G}$. Therefore, to satisfy the inter-controller latency requirement while adding new controllers to the current controller placement, the new controllers must be a subset of the same maximal clique (its nodes correspond to the nodes in $\boldsymbol{Q}$) from which the current controllers have been chosen. Moreover, it is assumed that we are given a maximum predefined increment parameter ($\gamma$) which indicates the maximum number of new controllers that can be added to the set of existing controllers. It is clear that the upper bound of $\gamma$ is $|\boldsymbol{Q}| - |\boldsymbol{C}|$. It should be noted that since we apply ICPP when the *switch reassignment* (using URSRP or BRSRP) is not viable and the ICPP itself involves *switch reassignment*, we consider the unbudgeted form of the *switch reassignment* for ICPP.

### 3.3.5 Hardness Analysis

It can be seen that BRSRP and ICPP are variants of URSRP. We analyze the NP-hardness of URSRP by mapping it to a well-known NP-hard problem. Thus, it is highly unlikely to have an efficient (i.e., with polynomial time complexity) optimal algorithm to solve URSRP.

**Theorem 1.** *URSRP is NP-hard.*

*Proof.* We show that URSRP can be mapped to the NP-hard problem of scheduling jobs on identical parallel machines with machine eligibility constraints and arbitrary processing sets [84, 86]. Consider $m$ identical machines which run in parallel and $n$ independent jobs where each job has a processing time. The makespan of a schedule is defined as the maximum of the total time used by any machine. The objective of such a problem is to seek a schedule that minimizes the makespan. It should be noted that *identical machines* mean that the processing time of a job is independent of the machine that processes it. Moreover, *machine eligibility* means that each job can be processed by only a subset of the machines, which is called the *processing set* of that job. Such a problem is denoted by $Pm|C_i|M_{\max}$, where $Pm$, $C_i$, and $M_{\max}$ indicate identical parallel machines, the processing set of a job $i$, and the makespan, respectively. In particular, URSRP with $r = 1$ is mapped to $Pm|C_i|M_{\max}$ with arbitrary processing sets. Arbitrary indicates that there can be any set relationship

(such as equality, having overlaps, subset or disjointness) between the processing sets of the jobs. Each controller is mapped to a machine. A flow (in aggregated form) from a switch is considered as a job and the traffic load of a switch (i.e., the demand of a flow) relates to the processing time of a job. The maximum load on a controller after switch reassignment corresponds to the makespan. Therefore, minimizing $\phi$ corresponds to the minimization of the makespan. It should be noted that having $r > 1$ ($r << |\boldsymbol{S}|$) makes the problem harder to solve since each switch must be assigned to more than 1 controller while satisfying the constraints. □

**Remark 1.** *Let $OPT$ be the optimal value of the URSRP and $OPT_\delta$ be the optimal value of the BRSRP. If we remove the reassignment cost constraint in (3.10), BRSRP reduces to URSRP. Thus, $OPT \leq OPT_\delta$ is true due to the fact that URSRP is a relaxed version of BRSRP.*

## 3.4 Proposed Solutions

A promising approach to solve NP-hard discrete optimization problems such as the *switch reassignment* is to find a solution that approximates the optimal solution with regard to its value. In the following, we provide approximation algorithms for solving the *switch reassignment* problem by transforming its variants to the problem of scheduling jobs on parallel machines with machine eligibility constraints.

### 3.4.1 SRBG Algorithm

To solve the URSRP, we propose a 2-approximation solution approach, called *Switch Reassignment with Bipartite Graph rounding* (SRBG), which is a modified version of the method based on matching in a bipartite graph to solve the problem of *scheduling unrelated parallel machines* in [89]. In such a problem, the machines are not identical and the processing time of job $i$ on machine $j$ is $l_{ij}$. To convert URSRP to the aforementioned problem, we define the load of switch $i$ on controller $j$ as follows.

$$l_{ij} = \begin{cases} l_i, & \forall j \in \boldsymbol{C_i} \\ \infty, & \text{otherwise} \end{cases} \tag{3.22}$$

After this modification, we form the corresponding relaxed Linear Programming (LP) problem. Then, we utilize the rounding technique in [89] to round a fractional solution

**Algorithm 4** SRBG

1: Find $\lambda^*$ as the minimum value of $\lambda$ such that $LP(\lambda)$ is feasible.
2: Find a vertex solution $\tilde{x}$ to $LP(\lambda^*)$.
3: Construct a bipartite graph $\boldsymbol{B}$ on vertices in the set $\boldsymbol{S'} \cup \boldsymbol{C}$ such that it has the edge $(i, j)$ if $\tilde{x}_{ij} \neq 0$.
4: If $\tilde{x}_{ij} = 1$, assign switch $i$ to controller $j$ and remove the corresponding edge from $\boldsymbol{B}$.
5: Name the remaining bipartite graph $\boldsymbol{H}$, which has been built on vertices in the set $\boldsymbol{F} \cup \boldsymbol{C}$, where $\boldsymbol{F}$ includes the fractionally assigned switches.
6: Find a matching $\boldsymbol{M}$ (which covers all the vertices in $\boldsymbol{F}$) in $\boldsymbol{H}$ and assign the switches to the controllers accordingly.

to the LP relaxation. In particular, a feasibility problem, called $LP(\lambda)$, is defined as follows.

$$\sum_{j:(i,j)\in T_\lambda} \tilde{x}_{ij} = 1, \quad \forall i \in \boldsymbol{S'} \tag{3.23}$$

$$\sum_{i:(i,j)\in T_\lambda} l_{ij}\tilde{x}_{ij} \leq \lambda, \quad \forall j \in \boldsymbol{C} \tag{3.24}$$

$$\tilde{x}_{ij} \geq 0, \quad \forall (i,j) \in T_\lambda. \tag{3.25}$$

Assume that we know the value of the objective function (denoted by $\lambda$) and $T_\lambda = \{(i,j) \mid i \in \boldsymbol{S'},\ j \in \boldsymbol{C},\ l_{ij} \leq \lambda \wedge \nexists (i_1, j),\ (i_2, j) \mid i_1 \in \boldsymbol{S_{m_1}},\ i_2 \in \boldsymbol{S_{m_2}},\ m_1 = m_2\}$, that is, the set of all assignments for which the traffic load of a switch on a controller is less than or equal to the value of $\lambda$ and no two switches from the same class are assigned to the same controller. Other variables corresponding to the possible assignment of a switch with traffic load more than $\lambda$ to a controller are removed. The next steps are summarized in *Algorithm* 4.

**Theorem 2.** *Algorithm 4 achieves a polynomial* 2*-approximation solution [89].*

*Proof.* For this chapter to be self-contained, we briefly summarize the idea behind the proof. However, we refer the interested readers to [89] for more details. First, it is proved that $\boldsymbol{B}$ is a pseudo-forest, i.e., each of its connected components has number of edges at most equal to the number of vertices (in other words each connected

component of $\boldsymbol{B}$ is a pseudo-tree). The reason is that a vertex solution to $LP(\lambda)$ has at most $|\boldsymbol{S'}| + |\boldsymbol{C}|$ non-zero variables and we know that the number of non-zero variables is equal to the number of edges in $\boldsymbol{B}$ (each edge of $\boldsymbol{B}$ corresponds to a non-zero variable after solving the LP relaxation). Thus, the number of edges is equal to the number of nodes ($|\boldsymbol{S'}|+|\boldsymbol{C}|$) and $\boldsymbol{B}$ is a pseudo forest. If $LP(\lambda)$ and $\tilde{x}$ are limited to the switches and controllers in a connected component of $\boldsymbol{B}$, a feasible solution to the new feasibility problem is also a vertex solution. Therefore, the number of vertices of the considered connected component is the total number of aforementioned switches and controllers, and the number of its edges is at most the same. Consequently, it results in a pseudo-tree. Additionally, since $\boldsymbol{H}$ is constructed by removing an equal number of edges and vertices from $\boldsymbol{B}$ (i.e., the removal of edges corresponding to the integral assignments), it is a pseudo-forest.

Second, it is shown that there is a matching $\boldsymbol{M}$ in $\boldsymbol{H}$ that covers all of the switches in it. The degree of every switch in $\boldsymbol{F}$ is at least 2 since it has a fractional assignment to at least 2 controllers. If $\boldsymbol{H}$ has leaves (vertices with degree 1), they are the controllers and the edges incident to such vertices are added to matching $\boldsymbol{M}$ and their endpoints are removed from $\boldsymbol{H}$. In addition, if $\boldsymbol{H}$ has cycles, a matching is found by including the alternate edges. Therefore, we will have a matching for $\boldsymbol{H}$ that covers all the switches.

Finally, after connecting all the non-fractionally assigned switches of $\boldsymbol{S'}$ to their associated controllers in step 4 of *Algorithm* 4, the maximum load on a controller is at most $\lambda^*$. Then, finding the matching in step 6 results in having at most one additional switch assigned to a controller. Assume $i$ is a fractionally assigned switch and it is matched to controller $j$. Due to the fact that $(i, j) \in T_{\lambda^*}$, $l_{ij}$ is at most $\lambda^*$. Therefore, the load on controller $j$ is increased by at most $\lambda^*$ after assigning a switch from the fractionally assigned switches. Thus, the objective value will be at most $2\lambda^*$. Moreover, since $\lambda^*$ is a lower bound for the optimal solution, *Algorithm* 4 gives a 2-approximation.

*Algorithm* 4 is polynomial in the size of input (i.e., $|\boldsymbol{S'}| \times |\boldsymbol{C}|$) [89]. We define $n = |\boldsymbol{S'}| \times |\boldsymbol{C}|$ and thus, we show the time complexity of the algorithm is in $O(n^\alpha)$ where $\alpha$ is a positive constant. All the steps including seeking $\lambda^*$ (using a binary search in step 1), finding a vertex solution, constructing the bipartite graph and finding a matching in the graph can be carried out in polynomial time. □

### 3.4.2 SDRBG Algorithm

To have an approximate solution to BRSRP and due to the fact that we have decomposed BRSRP into two sub-problems, we propose a two-phase algorithm called *Switch Dissociation and Reassignment with Bipartite Graph rounding* (SDRBG). The two phases are *Greedy Switch Dissociation* (GSD) and *Switch Reassignment* (SR). It should be noted that after running the two phases we carry out a feasibility check to ensure that the obtained solution is a feasible solution to BRSRP.

#### Solving the switch dissociation sub-problem

It should be noted that if a switch $i$ has no choice but to remain connected to its current controllers (i.e., $\boldsymbol{C_i} = \{c_i\}$), it must not be dissociated from any of its currently associated controllers. Therefore, as a *preprocessing* step before the dissociation, we identify such switches and exclude them from the list of candidates for dissociation. Moreover, in case of a change in the switch-controller latency requirement, some of the existing switches must be dissociated from their currently associated controllers due to the violation of the latency constraint. If the number of such switches is more than $\delta$, this *switch dissociation* problem instance becomes infeasible. We solve the *switch dissociation* sub-problem using a greedy optimal algorithm as follows.

As shown in *Algorithm 5*, we find the maximum loaded controller in the current switch-controller assignments (step 5). Then in the next step, the switch with the maximum load is dissociated from controller $c$. Next, both the load on controller $c$ and $\boldsymbol{A}$ (i.e., the set of switch-controller assignments) are updated. This process is repeated $\delta$ times. Finally, the solution including $\phi_j^-$ $(\forall j \in \boldsymbol{C})$ and $\boldsymbol{D}$ (the set of dissociated switches) is returned. It is worth mentioning that in case of having ties in choosing a controller of a switch, we randomly choose one. Note that if each switch has been assigned to $r > 1$ controllers in the current assignment, after the dissociation phase, it is possible that the switch is dissociated from one or all of its associated controllers. The time complexity of *Algorithm 5* is $O(\delta(|\boldsymbol{S'}| + |\boldsymbol{C}|))$ since step 5 and step 6 have $O(|\boldsymbol{C}|)$ and $O(|\boldsymbol{S'}|)$ complexity, respectively and they are repeated $\delta$ times.

**Lemma 1.** *The solution produced by the GSD algorithm is optimal.*

*Proof.* We prove the optimality of *GSD* algorithm by contradiction. Suppose $\boldsymbol{D}$ is not the optimal set of switches to dissociate from the controllers and its associated objective value is denoted by $\varphi$. Therefore, there exists a solution where $\varphi > \phi'$ and

**Algorithm 5** GSD

**Input**: $\boldsymbol{A}$, $\boldsymbol{L}$, $\delta$.
**Output**: $\boldsymbol{D}$, $\min \phi^-$, $\phi_j^-$, $\forall j \in \boldsymbol{C}$.
$k \leftarrow 1$
**while** $k \leq \delta$ **do**
    $c \leftarrow \text{argmax}_j \, \phi_j$.
    $s \leftarrow \text{argmax}_{i \in S_j} \, l_i$.
    $\phi_c \leftarrow \phi_c - l_s$.
    $\boldsymbol{A} \leftarrow \boldsymbol{A} \setminus (s, c)$.
    $k \leftarrow k + 1$.
**end while**

$\boldsymbol{D'}$ is the set of switches to be dissociated. However, we know that $|\boldsymbol{D}| = |\boldsymbol{D'}| = \delta$ and *GSD* always chooses the switch with the maximum traffic load from the most heavily loaded controller. Thus, the switches in both $\boldsymbol{D}$ and $\boldsymbol{D'}$ or their total load (in case of ties) must be the same which results in $\varphi = \phi'$. This contradicts our initial assumption of having $\varphi > \phi'$. □

**Lemma 2.** $\varphi \leq OPT_\delta$.

*Proof.* From Lemma 1, we know that $\boldsymbol{D}$ is the optimal set of switches to be dissociated from their controllers and $\varphi$ is its corresponding optimal objective value. It should be noted that $\boldsymbol{D}$ may not be the optimal set of switches to be dissociated for the BRSRP. Let $\boldsymbol{D^*}$ and $\varphi^*$ be the set of dissociated switches and the maximum load on the controllers after the dissociation of $\boldsymbol{D^*}$ switches in the optimal solution to BRSRP, respectively. Since the value of $\varphi$ is optimal (it is the lower bound of $\varphi^*$) and both $\boldsymbol{D}$ and $\boldsymbol{D^*}$ have $\delta$ switches, $\varphi \leq \varphi^*$ (i.e., the dissociation of the switches in $\boldsymbol{D^*}$ may result in a higher maximum load on the controllers). Moreover, $OPT_\delta$ is the maximum load on a controller after re-associating the switches in $\boldsymbol{D^*}$. As a result, we have $\varphi \leq \varphi^* \leq OPT_\delta$. □

### The solution to the switch reassignment sub-problem

We adopt a similar approach to [90] (which also has its roots in the LP relaxation and rounding in [89]) while we further take the resiliency into consideration. The LP relaxation of $P_{2-2}$ (shown in (3.17)) and its corresponding feasibility problem are denoted by *SR-LP* and *SR-FLP*, respectively.

**Algorithm 6** SR

1: **Input**: $\boldsymbol{D}$, $\boldsymbol{C}$, $\phi_j^-$ $(j \in \boldsymbol{C})$.
2: **Output**: $w^I$, $\phi_g^*$.
3: $lb = \phi_f$.
4: $ub =$ The objective value of a feasible solution.
5: **while** $ub > (1+\epsilon)lb$ **do**
6: $\quad \phi_g = (lb + ub)/2$.
7: $\quad$ Remove $j$ from $C_i$, if $l_{ij} >= \phi_g - \phi_j^-$, $\forall i \in \boldsymbol{D}$
8: $\quad$ **if** *SR-FLP*$(\phi_g)$ is feasible **then**
9: $\quad\quad ub = \phi_g$.
10: $\quad$ **else**
11: $\quad\quad lb = \phi_g$.
12: $\quad$ **end if**
13: **end while**
14: Find a fractional solution to *SRP-FLP*$(ub)$ and round it to a factor-2 integer solution $w^I$ similar to Section 3.4.1.

The objective value of any feasible solution to the modified *switch reassignment* sub-problem provides an upper bound for $\phi$. Such a feasible solution can be found by randomly reassigning switches to the controllers. To acquire a lower bound for the solution to the *switch reassignment*, we solve its corresponding relaxed LP problem. The resultant fractional optimal value (if any), denoted by $\phi_f$, serves as a lower bound for $\phi$. To seek the smallest value $\phi_g^*$ such that *SR-FLP*$(\phi_g)$ has a feasible solution, binary search is employed (lines 5−13 of *Algorithm 6*). At each iteration, we guess a value for the objective of the fractional solution $(\phi_g)$. Then, if *SR-FLP*$(\phi_g)$ has a feasible fractional solution, we set $\phi_g$ to the upper bound. Otherwise, it is set to the new lower bound. It should be noted that $\epsilon$ is used to tune the time complexity and usually has a small value close to 0. Finally, once the `while` loop is finished, the best feasible fractional solution (i.e., *SR-FLP*$(ub)$) is computed and then rounded to an integer solution. $\phi^*$ is the optimal objective value of the integer solution to the *switch reassignment* sub-problem. Therefore,

$$\sum_{i \in \boldsymbol{D}} l_i w_{ij}^I \leq 2(ub - \phi_j^-) \tag{3.26}$$

$$\leq 2((1+\epsilon)lb - \phi_j^-) \tag{3.27}$$

$$\leq 2((1+\epsilon)\phi_g^* - \phi_j^-) \tag{3.28}$$

$$\leq 2(1+\epsilon)\phi^* - 2\phi_j^-, \quad \forall j \in \boldsymbol{C}. \tag{3.29}$$

Note that (3.28) is due to the fact that $lb$ is set in *Algorithm 6* resulting from the infeasibility of the problem, and thus $lb < \phi_g^*$.

Regarding the time complexity, line 3 takes $O(1)$ while line 4 (finding a feasible solution) has a polynomial time complexity. The `while` loop iterates $\log_2(\frac{ub-lb}{\epsilon lb})$ times. Line 6 and line 7 take $O(1)$ and $O(|\boldsymbol{D}|)$ time, respectively. The feasibility check in lines 8−12 is performed in polynomial time, which is greater than the time complexity of other parts of the loop. Therefore, the time complexity of *Algorithm* 6 is polynomial in the size of input similar to *Algorithm* 4 and belongs to $O(n'^{\beta})$ where $n' = |\boldsymbol{D}| \times |\boldsymbol{C}|$ and $\beta$ is a positive constant.

**Theorem 3.** *The two-phase algorithm to solve BRSRP has an approximation ratio of* $(4+4\epsilon)$.

*Proof.* The details of the proof are as follows.

$$\phi = \max_j(\sum_{i\in\boldsymbol{D}} w_{ij}^I l_{ij} + \phi_j^-) \tag{3.30}$$

$$\leq \max_j(2(1+\epsilon)\phi^* - \phi_j^-) \tag{3.31}$$

$$\leq 2(1+\epsilon)\phi^* \tag{3.32}$$

$$\leq 2(1+\epsilon)\theta^* \tag{3.33}$$

$$\leq 2(1+\epsilon)\max_j(\sum_{i\in\boldsymbol{S'}} \tilde{x}_{ij}^* l_{ij} + (\phi_j^-)^*) \tag{3.34}$$

$$\leq 2(1+\epsilon)(\max_j \sum_{i\in\boldsymbol{S'}} \tilde{x}_{ij}^* l_{ij} + \max_j(\phi_j^-)^*) \tag{3.35}$$

$$\leq 2(1+\epsilon)(OPT+\varphi) \tag{3.36}$$

$$\leq 2(1+\epsilon)(OPT_{\delta}+\varphi) \tag{3.37}$$

$$\leq 2(1+\epsilon)(OPT_{\delta}+OPT_{\delta}) \tag{3.38}$$

$$\leq (4+4\epsilon)OPT_{\delta} \tag{3.39}$$

We know that (3.30) is true since $\phi$ is the maximum load on the controllers after applying the two phases of SDRBG (i.e., switch dissociation and switch reassignment). As we explained earlier, $\phi_j^-$ is the load on controller $j$ after the switch dissociation phase and $\sum_{i \in \boldsymbol{D}} w_{ij}^I l_{ij}$ is the added load on controller $j$ after the switch reassignment phase. Then, we obtain the relation in (3.31) using (3.29). Next, we define an auxiliary problem for the *switch reassignment* sub-problem with the difference that $\boldsymbol{D}$ is replaced with $\boldsymbol{S'}$ in the constraints in (3.19) and (3.20) (i.e., we consider reassignment of all the switches from all the classes). The objective value of the optimal solution to this auxiliary problem is denoted by $\theta^*$. Since $\boldsymbol{D} \subset \boldsymbol{S'}$, (3.33) and (3.34) are true (the left-hand side of the relation in (3.18) is increased by including all the switches which subsequently increases the objective value, i.e., $\phi^* \leq \theta^*$ since $\phi_j^-$ is fixed). Furthermore, it is clear that an optimal solution to URSRP (denoted by $\tilde{x}_{ij}^*$) is a feasible solution to this auxiliary problem. Thus, $\theta^*$ is replaced in relation (3.34) which subsequently lead to (3.35). Then, $\max_j \sum_{i \in \boldsymbol{S'}} \tilde{x}_{ij}^* l_{ij}$ is replaced with $OPT$ in (3.36) considering Remark 1. Also, since $\max_j(\phi_j^-)^*$ is the objective of the switch dissociation phase it can be replaced by $\varphi$ in (3.36). Then, we obtain (3.37) and (3.38) due to Remark 1 and Lemma 2, respectively. Subsequently, we acquire (3.39) from (3.38). It should be noted that since we assume $\epsilon$ is a small value close to 0, we can replace $4\epsilon$ with $\epsilon$ in (3.39). Since the *GSD* algorithm is optimal (proved in Lemma 1), the two-phase algorithm provides an approximation ratio of $(4+4\epsilon)$. □

**Algorithm 7** Incremental Controller Placement (ICP)

1: **Input**: $\boldsymbol{Q}$, $\boldsymbol{L}$, $\boldsymbol{A}$.
2: **Output**: $\phi$.
3: $i = 1$.
4: $\boldsymbol{C_{\text{init}}} = \boldsymbol{C}$.
5: $\boldsymbol{Q'} = \boldsymbol{Q} \setminus \boldsymbol{C}$.
6: **while** $i \leq \gamma$ **do**
7: $\quad \boldsymbol{C} = \boldsymbol{C_{\text{init}}}$.
8: $\quad j = i$.
9: $\quad$ **while** $j > 0$ **do**
10: $\quad\quad \boldsymbol{C} = \boldsymbol{C} \cup \{q\}$. ▷ $q \in sorted(rank(\boldsymbol{Q'}))$
11: $\quad\quad j = j - 1$.
12: $\quad$ **end while**
13: $\quad$ **if** *feasible*($P_1$) **then**
14: $\quad\quad$ *Return the solution to URSRP using Algorithm 4.*
15: $\quad$ **else**
16: $\quad\quad i = i + 1$.
17: $\quad$ **end if**
18: **end while**

### 3.4.3 The Solution to ICPP

To solve this problem, we can solve a URSRP with a modified input in terms of the existing controllers. As shown in *Algorithm* 7, up to $\gamma$ controllers from $\boldsymbol{Q} \setminus \boldsymbol{C}$ (as we mentioned earlier, $\boldsymbol{Q}$ is the set of controllers from which the current controllers have been chosen) are added to the set of existing controllers $\boldsymbol{C}$ (line 5). In particular, in each iteration, the number of controllers is increased by one. The controllers in $\boldsymbol{Q'}$ are ranked based on the number of switches that they can cover (with regard to $sc_{\max}$). Then, they are sorted in a decreasing order of their ranks. Among the controllers of the same rank, the one with the smallest total switch-controller latency is selected first.

We analyze the time complexity of *Algorithm* 7 as follows. While the third and fourth lines take $O(1)$, the fifth line takes $O(|\boldsymbol{Q}|)$. The outer `while` loop iterates $\gamma$ times which is also the maximum number of iterations of the inner `while` loop. The time complexity of the inner loop is $O(\gamma|\boldsymbol{Q'}|\log|\boldsymbol{Q'}|)$ whereas the time complexity of lines 13−17 is $O(n^\alpha)$ (as described in Section 3.4.1). Therefore, the time complexity of *Algorithm* 7 is $\max\{O(\gamma^2|\boldsymbol{Q'}|\log|\boldsymbol{Q'}|), O(\gamma n^\alpha)\}$.

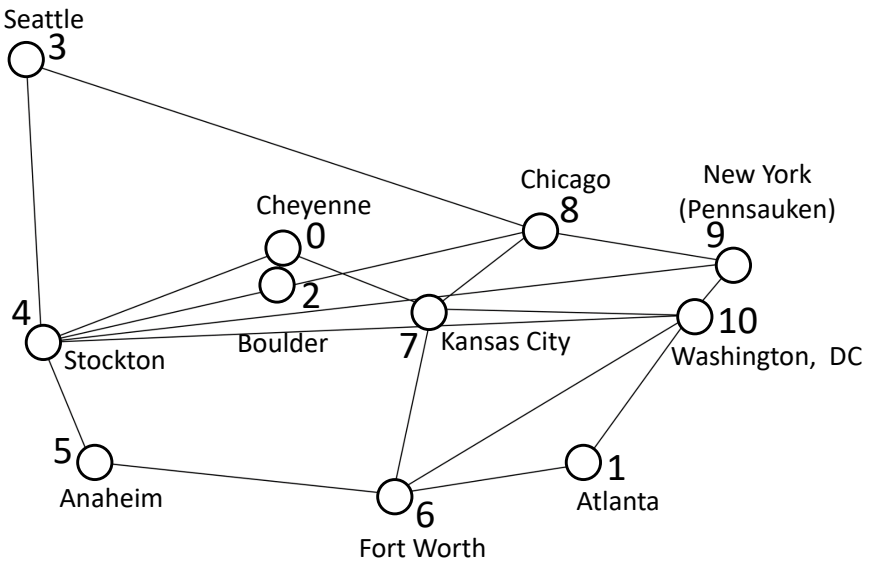


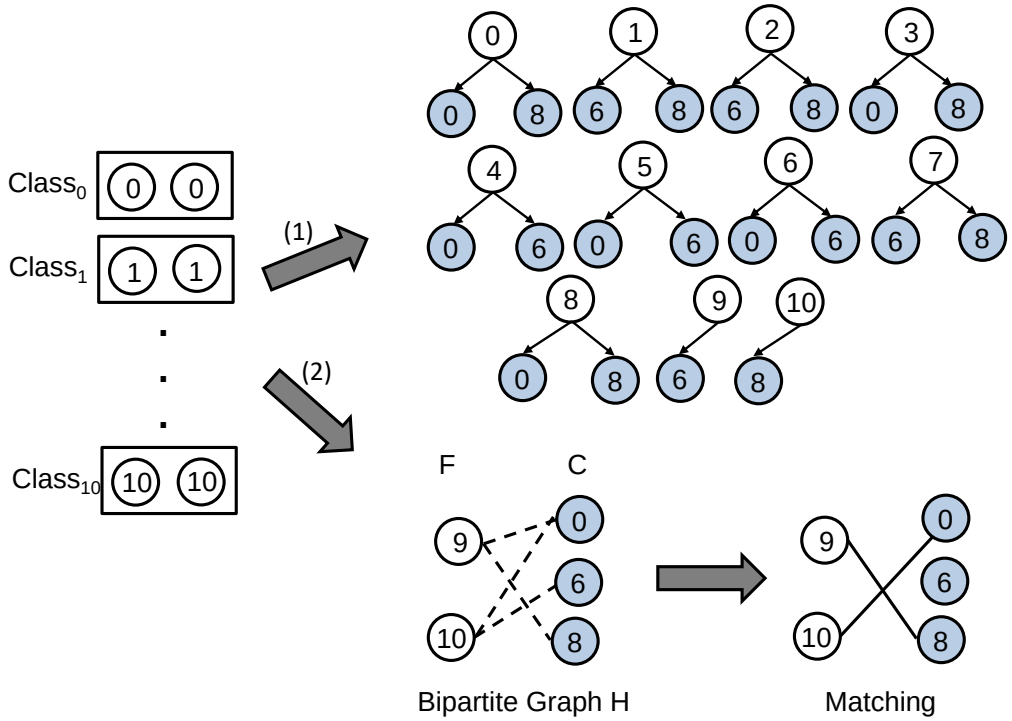

Figure 3.2: The Sprint topology.

### 3.4.4 Case Study

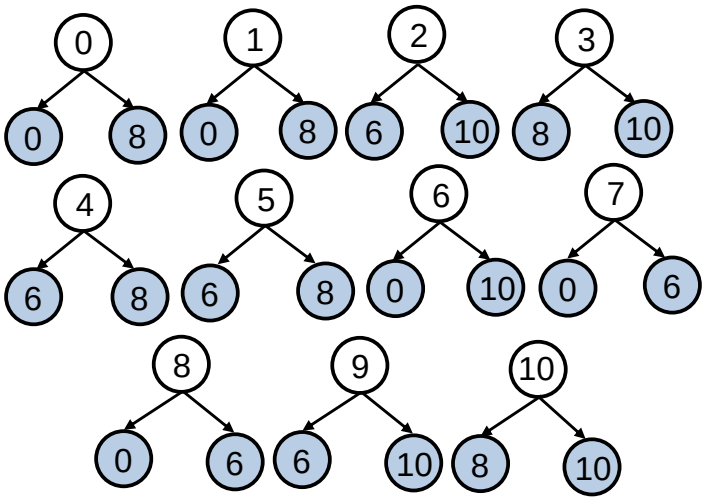
To better understand the working principles of the proposed algorithms, we provide a case study for the Sprint topology (shown in Figure 3.2). The values of inter-controller latency, switch-controller latency, the maximum capacity of a controller, and $r$ are set to 19.2 ms, 14.4 ms, 5000 kreq/s, and 2, respectively. We have chosen these values to simulate a case for a topology that would result in using all of our proposed algorithms. The initial traffic loads of switches are assumed to be 400 kreq/s and the controllers are placed at nodes $\{0, 6, 8\}$. Then, the traffic loads of switches increase uniformly by 40%. As a result, the maximum incurred load on the controllers is increased from 4000 kreq/s to 5600 kreq/s. In particular, the controllers at node 0 and node 8 are overloaded (the incurred loads are 5600 kreq/s and 5040 kreq/s, respectively). Therefore, the URSRP should be solved.

Solving this problem using the SRBG algorithm leads to the optimal maximum incurred load of 4480 kreq/s. Figure 3.3a depicts the resultant assignment after LP relaxation. Since $r > 1$, we create 11 classes with regard to the nodes of the graph. While the first step (1) shows the assignments corresponding to $\tilde{x}_{ij} = 1$, the second step (2) illustrates the rounding procedure of the fractional assignments using the bipartite graph and matching. The white nodes indicate the switches whereas the blue nodes determine the controllers. For instance, if a white node with ID 7 is

connected to the two blue nodes with ID 6 and ID 8, it means that a switch at node 7 is assigned to two controllers at node 6 and node 8.




(a) URSRP.



(b) ICPP.

Figure 3.3: Finding approximate solutions for the case study.

Assume we intend to solve the same problem but using BRSRP with the cost of $0.2|\boldsymbol{A}|$, i.e., only 20% of the current assignments can be changed to balance the load among the existing controllers. As mentioned in Section 3.4.2, we first determine which switches cannot be dissociated from their currently assigned controllers. In this example, the set of such switches is $\{1, 3, 4, 5\}$. Then, three switches (corresponding to five assignments, i.e., $\lceil 0.2 \times 22 \rceil$) $\{0, 2, 10\}$ are dissociated from their controllers using *Algorithm* 5. In particular, switch 0 and switch 10 are dissociated from both of their assigned controllers while switch 2 is only dissociated from one of its assigned controllers.

Next, we need to solve the SR sub-problem to reassign the dissociated switches to new controllers in order to achieve a better load balance among the controllers. In the optimal solution to the SR sub-problem, the maximum load on the controllers is 4480 kreq/s (we see in the following that the distribution of the load among the controllers will be different in the sub-optimal solution). Assume we cannot obtain such an optimal solution (which is the case for large problem sizes), thus using *Algorithm* 6 gives a sub-optimal solution to the SR sub-problem. This leads to the violation of the controller capacity for the controller at node 0 (5040 kreq/s). Consequently, we must solve the ICPP. This leads to $\{0, 6, 8, 10\}$ (a controller at node 10 is added to the existing controllers) as the set of controllers and the corresponding URSRP is feasible. In particular, $\boldsymbol{Q'} = \{1, 2, 7, 9, 10\}$ (all the potential controllers in this set satisfy the inter-controller latency requirement) and $\gamma = 5$. This results in the maximum load on the controllers to be 3360 kreq/s.

## 3.5 Performance Evaluation

In this section, we first elaborate on our experiment setup and then we assess the performance of our proposed algorithms considering different metrics and parameters.

### 3.5.1 Experiment Setup

**Dataset:** Similar to Chapter 2, we used the ITZ dataset to carry out our experiments. In particular, a number of WAN topologies, including Sprint, AT&T, AARNET, GEANT (2012), ULAKNET, and TATA were chosen such that they cover different sizes and shapes (mesh, hub-and-spoke, and linear) of a network. More details about these topologies can be found in Table 3.2. We also conducted a pre-

Table 3.2: Information about the chosen network topologies

| | **Sprint** | **AARNET** | **AT&T** | **GEANT** | **ULAKNET** | **TATA** |
|---|---|---|---|---|---|---|
| Network size | 11 | 19 | 25 | 37 | 76 | 143 |
| Links | 18 | 24 | 56 | 58 | 76 | 181 |
| Diameter (ms) | 24.11 | 31.05 | 24.44 | 28.40 | 9.81 | 17.35 |
| Average node degree | 3.27 | 2.53 | 4.48 | 3.14 | 2 | 2.53 |
| Shape | mesh | linear | mesh | mesh | hub-and-spoke | mesh |

processing on the topologies similar to Chapter 2.

**Scenarios and assumptions:** We considered a set of scenarios, which are the trigger points for the *switch reassignment* in our framework (shown in Figure 3.1). In the first scenario, **UNI-DEC**, we assumed that the traffic loads of switches have a decreasing trend in a uniform manner. The second scenario, **UNI-INC**, involved an opposite trend in the traffic loads of switches (a uniform increase). While the former can represent the case in which the traffic load of the network reduces during a specific time/day, the latter can correspond to the growth of the global Internet traffic over the time [93].

In the third scenario, **NON-UNI**, we assumed that the traffic loads of switches change in a non-uniform manner, which is more realistic to represent the random incurred load by different switches at various times. It should be noted that we supposed that security measures are already in place in the network (e.g., blocking/capping the traffic from possibly compromised switches that overload the network unexpectedly). Therefore, the change in the traffic loads of switches (especially the increase) does not result from a Denial of Service (DoS) or similar attacks. Nevertheless, load balancing would be useful to alleviate the impact of such security attacks to some extent. Finally, in the fourth scenario (**SC-CHG**), we consider the case where a QoS requirement in the network changes. In particular, the value of $sc_{\max}$ is reduced to reflect a tighter latency bound between a switch and its associated controllers. Moreover, similar to [39], we have assumed that our SDN-based network is reactive, which represents the worst-case scenario. That is, each flow receives its routing rule from the controller.

**Metrics:** Our main metric of interest is the maximum load on a controller in case of a uniform scenario and its average value in a non-uniform scenario. Using SRBG and SDRBG algorithms, such a load is adjusted with regard to the change in the traffic loads of switches. Another metric of interest is the execution time of the algorithms.

Table 3.3: Parameter settings

| Parameter | Value |
|---|---|
| $\delta$ | $[0.1, 0.9] \times \lvert A \rvert$ (with step size of 0.1) |
| r | {1, 2} |
| $cc_{\max}$ | 30 ms |
| $sc_{\max}$ | 25 ms |
| $\gamma$ | 6 |
| $\epsilon$ | 0.001 |
| Number of experiments (NON-UNI scenario) | 30 |
| Controller capacity | 5,000 kreq/s |
| Initial switch traffic loads | 400 kreq/s |
| Switch traffic loads (NON-UNI scenario) | [200, 600] kreq/s |
| Increase/decrease rate of switch traffic loads (UNI-INC/DEC scenario) | 20% |

**Setting the parameters:** Table 3.3 shows the assigned values to different parameters of URSRP, BRSRP, and ICPP in our experiments. For each topology, we assume that we are given a controller placement solution that satisfies $sc_{\max} = 25$ ms and the maximum controller capacity of 5000 kreq/s. The applied values for the traffic loads of switches and controller capacity are based on prior studies on the capacitated CPP [25, 30, 39, 45] as well as research conducted on the performance of SDN controllers [75, 76]. Moreover, according to Cisco Visual Networking Index (VNI) report [94], the IP traffic is expected to have an annual growth of 22% from 2015 to 2020. Therefore, the uniform increasing and decreasing rates of switch traffic loads are set to 20% (we assumed that this change in traffic volume corresponds to the change in the number of flow setup requests from the switches). If URSRP or BRSRP are infeasible with regard to the current parameters of the network or running SRBG and SDRBG algorithms results in the violation of the available controller capacity at a geographical location, the ICPP is solved using *Algorithm* 7. The value of $sc_{\max}$ is 25 ms which corresponds to the required 50 ms round trip propagation latency between the switch and each of its associated controllers in ISP networks [11]. However, it can be set to a lower value and we will analyze the impact of such a decrease in the following section (in the **SC-CHG** scenario, the value of $sc_{\max}$ is reduced to 20 ms). It should be noted that since $sc_{\max}$ is approximated by the propagation latency in SD-WANs, its maximum value will be equal to the graph diameter. Thus, any chosen value can be represented as a percentage of the graph diameter (although using a threshold-based approach is more practical). The same applies to $cc_{\max}$.

**Experiments platform (software/hardware):** We carried out all the experiments on an Intel Core 790 i7-3770 CPU @3.40GHz and 32GB RAM with Windows 10 Pro (64-bit) installed. We developed our proposed algorithms using Python while the

optimal solutions for topologies with size lower than 30 were obtained using Gurobi optimization software [77] (version 7.5.2).

### 3.5.2 Results and Discussion

In this section, we first provide a discussion on the comparison of our proposed schemes with the existing work. Then, we analyze the results of *switch reassignment* for different scenarios with regard to the maximum load on the controllers (as the main metric) as well as the execution time. We also show for which topologies and parameters, we need to solve the ICPP.

**Comparison with the existing work:** It should be noted that we have provided approximation algorithms with guaranteed bounds with regard to the optimal solution (which is the best reference for comparison) while other existing work such as [35, 39] provides heuristics to solve the switch reassignment/dynamic controller placement problem with different objective functions and considering no reassignment cost and no resiliency (more information is given in Section 3.2). Therefore, in order to make a comparison between our scheme and [39] (LiDy+), we did the following. We used the controller placement algorithm of LiDy+ for switch reassignment, which is based on finding the smallest enclosing disk with regard to the switch-controller latency bound with restricted search (i.e., the set of potential locations for controllers is a subset of the switches). In particular, for each topology, starting from the same set of controllers and the same maximum controller capacity, using LiDy+ to adjust the controller placement results in a higher maximum load on a controller and a higher load imbalance for most of the topologies compared with SRBG ($r = 1$) in the **UNI-INC** scenario. Also, by increasing the topology size, the obtained maximum load on a controller by LiDy+ rises. This is due to the fact that LiDy+ minimizes the switch-controller latency without considering its impact on the load on the controllers. Table 3.4 shows the details of the comparison.

**The maximum load on the controllers:** Figure 3.4 shows the initial maximum load on the controllers which are later affected by triggering the SRBG, SDRBG, and ICP algorithms. In particular, it illustrates the initial maximum controller load for each topology (i.e., **INIT**) as well as showing the maximum load on the controllers after the change in the traffic loads of switches and before the *switch reassignment* in the **UNI-DEC**, **UNI-INC**, and **NON-UNI** scenarios. It should be noted that we show the loads for $r = 2$ since all topologies have a similar initial maximum load

Table 3.4: Comparison between SRBG and LiDy+ with regard to MAX-LOAD (kreq/s)

| Topology | Set of controllers | SRBG | LiDy+ |
|---|---|---|---|
| Sprint | {0} | 5280 | 5280 |
| AARNET | {0, 8} | 4800 | 8640 |
| AT&T | {5, 6, 7} | 4320 | 7680 |
| GEANT | {0, 9, 21, 35} | 4800 | 7680 |
| ULAKNET | {73–79} | 5280 | 26880 |
| TATA | {9, 89–99} | 5760 | 26880 |

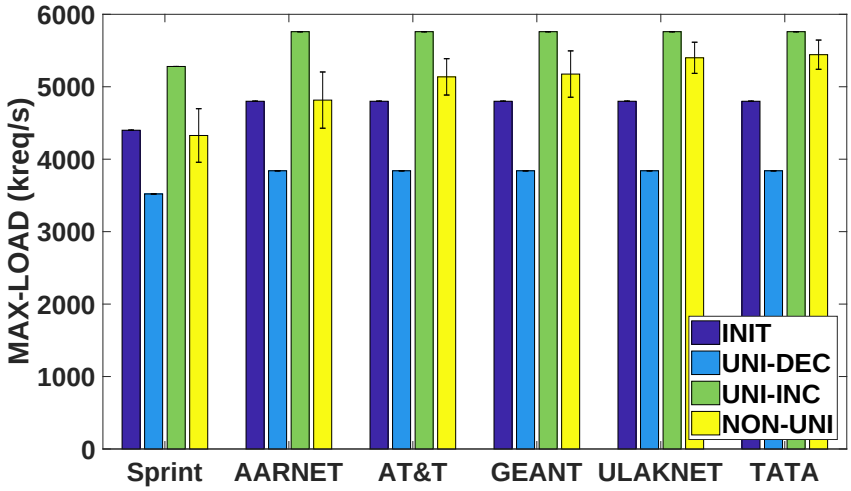



Figure 3.4: The change in the maximum load on the controllers in different scenarios (r=2).

Table 3.5: The number of controllers in the initial controller placement for different topologies

| Topology | r=1 | r=2 |
|---|---|---|
| Sprint | 1 | 2 |
| AARNET | 2 | 4 |
| AT&T | 3 | 5 |
| GEANT | 4 | 7 |
| ULAKNET | 7 | 13 |
| TATA | 12 | 24 |

on the controllers for $r = 1$ and 2 (although the number of controllers is different in the initial controller placement for both cases). Table 3.5 shows the initial number of controllers for different topologies.

Table 3.6: MAX-LOAD (kreq/s) of the SRBG algorithm in the UNI-DEC scenario and its corresponding budget value using the SDRBG algorithm

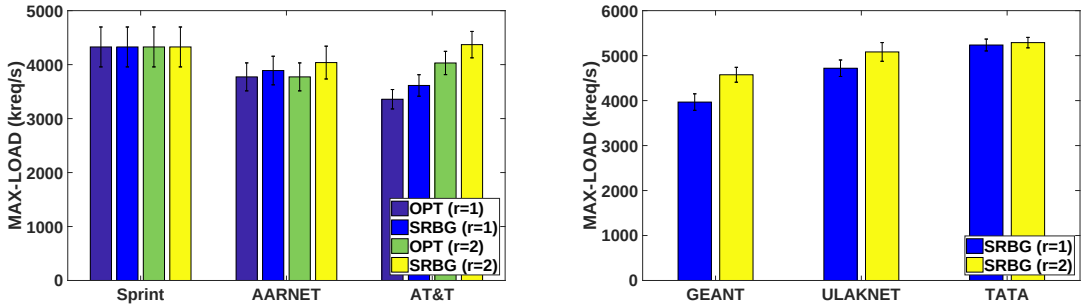
| **Topology** | **SRBG (r=1)** | **$\delta$ (r=1)** | **SRBG (r=2)** | **$\delta$ (r=2)** |
|---|---|---|---|---|
| Sprint | 3520 | $0.1\|A\|$ | 3520 | $0.1\|A\|$ |
| AARNET | 3200 | $0.1\|A\|$ | 3200 | $0.2\|A\|$ |
| AT&T | 2880 | $0.4\|A\|$ | 3200 | $0.2\|A\|$ |
| GEANT | 3200 | $0.2\|A\|$ | 3520 | $0.2\|A\|$ |
| ULAKNET | 3520 | $0.2\|A\|$ | 3840 | $0.1\|A\|$ |
| TATA | 3840 | $0.1\|A\|$ | 3840 | $0.1\|A\|$ |

Table 3.7: MAX-LOAD (kreq/s) of the SRBG algorithm in the UNI-INC scenario and its corresponding budget value using the SDRBG algorithm

| **Topology** | **SRBG (r=1)** | **$\delta$ (r=1)** | **SRBG (r=2)** | **$\delta$ (r=2)** |
|---|---|---|---|---|
| Sprint | 5280 | $0.1\|A\|$ | 5280 | $0.1\|A\|$ |
| AARNET | 4800 | $0.1\|A\|$ | 4800 | $0.2\|A\|$ |
| AT&T | 4320 | $0.3\|A\|$ | 4800 | $0.2\|A\|$ |
| GEANT | 4800 | $0.2\|A\|$ | 5280 | $0.2\|A\|$ |
| ULAKNET | 5280 | $0.2\|A\|$ | 5760 | $0.1\|A\|$ |
| TATA | 5760 | $0.1\|A\|$ | 5760 | $0.1\|A\|$ |

Table 3.6 and Table 3.7 summarize the acquired results from the **UNI-DEC** and **UNI-INC** scenarios, respectively. In particular, the second and fourth columns of both tables show MAX-LOAD (the maximum load on the controllers), which corresponds to the value of the objective function for URSRP with regard to different resilience levels using the SRBG algorithm. Moreover, the third and fifth columns provide the reassignmnet budget that leads to having the MAX-LOAD equal to the values shown in the second and fourth columns. Therefore, further increasing the value of $\delta$ does not reduce the MAX-LOAD. For example, considering the AT&T topology and Table 3.6, the same value of MAX-LOAD for solving URSRP can be achieved if we limit our budget (i.e., the allowed number of reassignments) to 20% of the total number of the current assignments for $r = 2$. It is worth mentioning that for small-size topologies such as Sprint and AARNET, the obtained MAX-LOAD in both of the uniform scenarios is almost equal to the optimal value.

Another interesting point is that for some of the topologies such as TATA, triggering the SRBG or SDRBG algorithms does not lead to a decreasing trend in MAX-LOAD (we compare the values of Table 3.7 with their corresponding initial values




(a) (b)

Figure 3.5: The average of the maximum load on the controllers for different topologies in the NON-UNI scenario by solving URSRP.

in Figure 3.4). In this case, the usage of the ICP algorithm is unavoidable unless it is viable for the network operator to increase the maximum controller capacity instead. Figure 3.5 illustrates the average of the maximum load on the controllers for different topologies in the **NON-UNI** scenario whereas Figure 3.6 depicts the results for metric with regard to different values of $\delta$ (here, we only show the results for $\delta \in \{\{0.1, 0.3, 0.5, 0.7, 0.9\} \times |A|\}$ to save space as well as due to the fact that there is no conspicuous change in the results for other values of $\delta$). Moreover, for topologies with smaller size (i.e., number of nodes), including Sprint, AARNET, and AT&T, the optimal value of the objective function (OPT) is shown as well (for larger topologies the Gurobi optimizer was unable to obtain the optimal results in a reasonable amount of time).

Considering Figure 3.5a and Figure 3.6a, for the Sprint topology regardless of the resilience level and the change in the traffic loads of switches, the average maximum load on the controllers stays the same. That is, even with the minimum set budget constraint (i.e., $\delta = 0.1|\boldsymbol{A}|$), MAX-LOAD does not change. This is due to the fact that for $r = 1$ there is only one controller at node 0 whereas for $r = 2$ the controllers are at nodes $\{0, 8\}$ and these are the minimum possible number of controllers to satisfy the required resilience. Hence, the assignment of the switches to the controllers has not been changed. Moreover, as shown in Figure 3.6, we can see that increasing the budget ($\delta$) does not necessarily lead to a decrease in the MAX-LOAD. It usually reaches a steady value which is almost equal to its peer for the unbudgeted case (in other words, solving the problem with unlimited budget gives a lower bound for the budgeted case) in Figure 3.5. For example, for the AT&T topology in Figure 3.6c

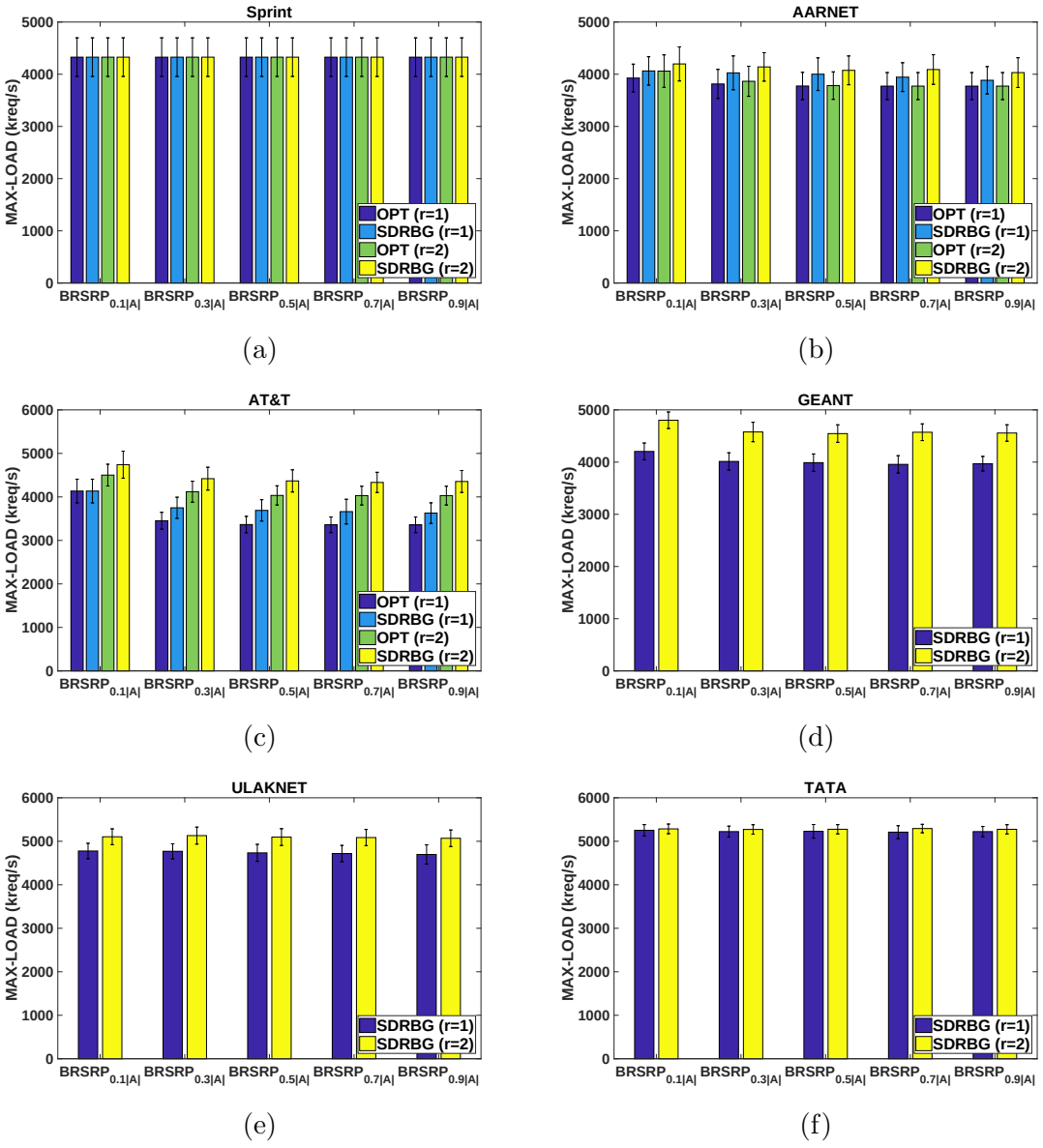



Figure 3.6: The average of the maximum load on the controllers for different topologies in the NON-UNI scenario through the solution of BRSRP.

Table 3.8: MAX-LOAD (kreq/s) of the SRBG algorithm in the SC-CHG scenario and its corresponding budget value using the SDRBG algorithm

| **Topology** | **SRBG (r=1)** | **$\delta$ (r=1)** | **SRBG (r=2)** | **$\delta$ (r=2)** |
|---|---|---|---|---|
| Sprint | 4400 | $0.1\|A\|$ | 4400 | $0.1\|A\|$ |
| GEANT | 4000 | $0.2\|A\|$ | 4400 | $0.2\|A\|$ |
| ULAKNET | 4400 | $0.2\|A\|$ | 4800 | $0.1\|A\|$ |
| TATA | 4800 | $0.1\|A\|$ | 4800 | $0.1\|A\|$ |

with $r = 2$ and $\delta > 0.5|\boldsymbol{A}|$, the optimal MAX-LOAD is 4030 kreq/s which is equal to the unbudgeted case in Figure 3.5. Such insights assist the network operators in achieving the same level of load balancing on the controllers with the minimum budget.

Using Figure 3.5 and Figure 3.6, we can answer the question of whether *there is a reassignment budget ($\delta$) that does not lead to the violation of the controller capacity for a given topology?* For instance, while for the GEANT topology solving both URSRP and BRSRP (with different budgets) regardless of the resilience level always keeps the MAX-LOAD below 5000 kreq/s, for TATA, it always results in the violation of the controller capacity in the considered **NON-UNI** scenario, which is similar to the uniform scenarios.

Table 3.8 summarizes the obtained results from the **SC-CHG** scenario. It can be seen that by limiting the budget to 20% change in the number of reassignments for the SDRBG algorithm, the same MAX-LOAD as the SRGB algorithm can be achieved. It should be noted that reducing $sc_{\max}$ to 20 ms makes the URSRP and BRSRP infeasible for AARNET and AT&T.

Furthermore, we measured Jain's Fairness Index (JFI) [95] as an indicator of the degree of fairness. JFI quantifies the equality of the distribution of the load to the controllers for different scenarios we measured. Therefore, if all the controllers have the same amount of incurred load by their connected switches, then the fairness index will be 1, which indicates the load balancing is 100% fair. The SRBG algorithm leads to having JFI of more than 0.99. A similar trend applies to the obtained results from the SDRBG algorithm with a budget that gives the same MAX-LOAD as SRBG.

**The execution time:** Figure 3.7 depicts the average execution time of the largest-size topology, i.e., TATA for SRBG and SDRBG algorithms in the **NON-UNI** scenario. The x-axis shows the SDRBG with regard to different values of $\delta$ as well as the SRBG algorithm. It can be seen that by increasing the reassignment

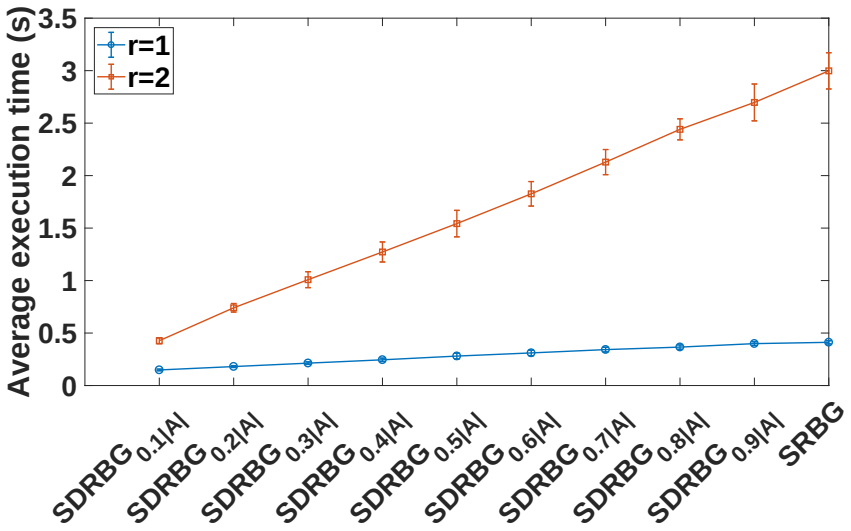



Figure 3.7: Average execution time of the SRBG and SDRBG algorithms for TATA in the NON-UNI scenario.

budget (i.e., $\delta$), the average execution time rises and it reaches its maximum value for the SRBG algorithm where there is no budget constraint. Also, if using the SDRBG algorithm with a budget, we can obtain the same MAX-LOAD achieved by the SRBG algorithm (or a close value to it), the execution time of the SDRBG is less than that of SRBG (usually in order of hundreds of milliseconds). Moreover, increasing the resilience level leads to the rise of the average execution time of both algorithms. The results for other topologies have a similar trend.

**Using the ICP algorithm:** As we mentioned in the previous sections, we can solve an ICPP using the ICP algorithm if obtaining a reassignment solution is not viable for any of the scenarios or they result in the violation of the controller capacity. It should be noted that the latter case does not necessarily mean that the optimal solution (if can be found in a reasonable amount of time given sufficient resources) also leads to the violation of the controller capacity. The obtained results from our algorithms are within the approximation bound as explained in Section 3.4. By looking back at Figure 3.5 and Figure 3.6, we can see that the violation of controller capacity happened for the larger size topologies, including ULAKNET and TATA. Additionally, as we mentioned earlier, using the **SC-CHG** scenario, it was not possible to find a feasible solution for the AT&T and AARNET topologies (however, reducing $sc_{\max}$ to 21 ms will result in feasible solutions for both topologies). Since the inter-controller latency bound ($cc_{\max}$) at the initial controller placement was 30 ms, which is more

Table 3.9: Set of controllers and the change in MAX-LOAD before and after using *Algorithm* 7 (r=2)

| Topology | Initial set of controllers | Controller(s) added | Change in MAX-LOAD |
|---|---|---|---|
| AARNET | {0, 3, 8, 9} | {10} | 4800 kreq/s → 3200 kreq/s |
| AT&T | {1, 4, 6, 15, 19} | {11} | 4800 kreq/s → 3600 kreq/s |
| ULAKNET | {7, 62, 64, 68, 69, 70, 71, 73–78} | {28, 54, 56} | 5760 kreq/s → 4800 kreq/s |
| TATA | {7, 8, 64–69, 72–80, 82–86, 88, 89} | {130, 131–133, 137} | 5760 kreq/s → 4800 kreq/s |

than the diameter of the aforementioned topologies, all the switch locations can serve as the potential locations for the controllers ($\boldsymbol{Q} = \boldsymbol{S}$). However, the service provider can have any preference to choose from only a subset of such potential locations.

Table 3.9 shows the obtained results from the ICP algorithm for AARNET and AT&T in the **SC-CHG** scenario as well as ULAKNET and TATA in the **UNI-INC** scenario with $r = 2$. While adding new controllers to the set of existing controllers for AARNET and AT&T is due to the change in the value of $sc_{\max}$ from 25 ms to 20 ms, adding new controllers to the set of existing controllers for ULAKNET and TATA stems from the violation of the controller capacity. It should be noted that while the maximum value of $\gamma$ is $|\boldsymbol{Q}'|$, we set it to 6. This value can be chosen according to the preference of the network operator, network requirements and the structure of the network topology.

## 3.6 Conclusions

In this chapter, we investigated the switch reassignment and incremental controller placement in SD-WANs. This is an important problem due to the dynamics of such networks in terms of both the traffic loads of switches and the new QoS requirements such as a new switch-controller latency bound. We provided approximate solutions to a set of optimization problems with/without a budget for resilient switch reassignment as well as the incremental deployment problem of the controllers. Such insights help network operators achieve cost-effective solutions while adapting their networks to dynamics and new requirements.

# Chapter 4

# Conclusions and Future Work

## 4.1 Conclusions

By bringing the benefits of SDN technology to legacy WANs, SD-WANs are one of the promising network architectures. The importance of SD-WANs stems from their abstraction of software from hardware, centralized and simplified architecture, ease of management, being more elastic, and reducing recurring network costs. In this dissertation, we have investigated the resiliency of the control plane in SD-WANs, which is of great significance. By separating the control plane from the data plane and centralized management, the correct functioning of SD-WANs is more reliant on the control plane. Therefore, proper mechanisms to improve the resiliency of the control plane should be in place not only at the design stage of SD-WANs but also during their lifespan. Such resilience mechanisms should be well-adapted to the dynamics and new requirements of SD-WANs over time. The aforementioned issues are modeled as discrete optimization problems that are NP-hard. Thus, we have proposed systematic and well-suited algorithmic approaches to solve these problems. In particular, this dissertation contains the following.

- **The design of a resilient control plane.** We proposed the design of a resilient control plane by focusing on the problem of resilient controller placement in SD-WANs. Such a problem aims to minimize the number of required controllers and involves the assignment of multiple controllers to each switch to prevent controller node failures while considering the capacity of controllers, the traffic loads of switches, and QoS constraints. The formulated optimization problem was solved using an algorithmic approach based on clique graphs. The

performance of the proposed algorithms was assessed using real datasets. The details of our proposed schemes were discussed in Chapter 2.

- **Resilient switch reassignment and incremental controller placement.** Having a static resilient control plane does not suffice to provide a tailored and continuous resilience support for SD-WANs. Due to the high dynamics of such networks and new QoS requirements that need to be fulfilled during their life span, proper adjustments must be made to the resilient controller placement. The aforementioned adjustments can be in the form of changing the assignments of switches to the controllers and/or incorporating more controllers into the controller placement to maintain resiliency as well as satisfying new requirements resulting from network dynamics. In Chapter 3, we modeled this problem as a variant of scheduling on parallel machines with processing set restrictions. Then, approximation algorithms were provided to solve the problem in polynomial time. Finally, the proposed algorithms were evaluated on real datasets.

Finally, as discussed in Appendix A, although SD-WANs have outstanding merits, due to the high initial cost of complete migration to such architectures as well as their associated technical challenges, an incremental migration is a viable choice for some service providers. This involves the progressive replacement of legacy routers with SDN-enabled devices in multiple time periods such that the budget constraints are satisfied and traffic engineering flexibility in the data plane is maximized. In most existing work, such traffic engineering flexibility in the data plane is characterized by the total number of alternative paths that are enabled through the migration of a number of legacy nodes to SDN-enabled devices. The alternative paths assist in providing rerouting options in the event of node/link failures in the data plane, which result in an efficient response to network dynamics. Therefore, we have investigated this problem as well and have included our research outcome in Appendix A.

## 4.2 Future Work

Regarding future work and plans beyond this dissertation, in the following, we discuss some ideas and potential research directions that would be beneficial to investigate.

In the short run, the proposed resilient controller placement solutions can be amended to cover both node and link failures in the control plane considering different

objective functions (e.g., minimizing the cost of controller deployment at different locations or minimizing the expected control path loss) with regard to the preference of service providers. Moreover, considering more constraints for switch reassignment such as switch priority (defined based on the preference of the service provider) would add extra benefits to the proposed model in this dissertation.

In the long run, the design of efficient randomized/approximation algorithms for the static resilient controller placement as well as improving the proposed approximation bounds for switch reassignment can be explored. Furthermore, defining proper and more explicit resiliency metrics for the data plane is an interesting topic for further research. Another area of research which is relevant to resiliency is the security of the SDN control plane. Security issues such as software failure, software alteration, and compromised controller(s) are more detrimental to SDN-based networks due to their reliance on the (logically) centralized control plane. Last but not least, similar issues to the problems investigated in this dissertation can be explored and addressed in the context of SDN-based wireless networks, which are more complex due to their highly dynamic nature that involves more constraints.

# Appendix A

# Enhancing Traffic Engineering Flexibility in the Data Plane by Incremental Migration to SDN

## A.1 Background

In this dissertation we mainly investigated the resilience of the SDN control plane, as the key building block of an SDN-based network. We have provided solutions to the *resilient controller placement problem* at the network design phase as well as *switch reassignment* and *incremental controller placement* while maintaining the resilience during the lifespan of an SD-WAN. In the aforementioned cases, the assumption is that all the switches in the network are SDN-enabled and we intend to have a resilient controller placement that can be adjusted over time with regard to the dynamics of the network (e.g., the change in the traffic loads of switches).

While some network operators such as Google have already deployed SD-WANs, others prefer an *incremental deployment* of SDN in their networks due to its huge operational and performance costs as well as organizational and technical challenges, especially in large-scale networks [93,96]. This leads to a hybrid network that consists of legacy routers and SDN-enabled devices before a full migration to SDN. Obviously, the service providers intend to maximize the benefits that they gain from an *incremental deployment* of SDN in their network. Considering resilience, it is still possible to provide a resilient control plane for the part of the network that is SDN-enabled. However, the challenge is how to choose the legacy network devices to be upgraded

over time such that traffic engineering flexibility can be improved while taking into account the available budget for migration. Enhancing traffic engineering flexibility subsequently improves the resilience of the data plane. In particular, the centralized, fine-grained, and per-flow SDN-based routing opens avenues to facilitate an instant response to network dynamics such as link/node failures or congestion [93] in the data plane. This is in contrast with conventional protocols such as Open Shortest Path First (OSPF) that provide inflexible destination-based routing (mostly based on the shortest path).

It should be noted that the aforementioned upgrade can be achieved by incorporating hybrid control plane architectures into the network devices (hybrid equipment), or replacing the legacy routers with OpenFlow switches. The former poses various challenges to the service providers with regard to network management. The main issue is that one scheme (SDN operation/legacy routing) is oblivious to the configurations made by the other one (legacy routing/SDN operation). This may result in an unsolicited increase in the size of forwarding tables, routing loops or black holes in case of network failures [97]. Hence, a more practical approach is to replace the legacy IP routers with fully SDN-enabled equipment.

As mentioned earlier, due to the high cost and technical challenges of a full migration to SDN, some network operators prefer an *incremental/progressive migration*, i.e., upgrading a subset of legacy network devices in an existing network to SDN within a time span of years (or months). This raises the question of how many legacy routers and in which sequence they should be upgraded to maximize the resilience benefits in the data plane while satisfying the budget and other possible resource constraints. The upgrade cost of the legacy devices can reflect their heterogeneity in terms of their role in the network (i.e., serving as a core or edge switch) and their remaining lifetime. Moreover, the gradual decrease in the cost of SDN deployment as a new technology would encourage ISPs to favor not-so-early migrations [93]. Thus, there is a trade-off between the performance gain from enhancing resiliency by early migration and the investment.

In this research, we investigate the problem of finding an appropriate migration sequence for the legacy nodes of a given network with regard to the total budget and device replacement cost over a multi-period time span more thoroughly compared with the most related research [93, 98] (other work such as [96, 99] considered a different objective function for a single time period). Our contributions are twofold. First, we propose a greedy algorithm that achieves solutions in par with the state-of-the-art

for small-scale topologies as well as having a higher solution quality especially for large-scale topologies. Second, we propose a metaheuristic algorithm that not only improves the solution quality of our greedy algorithm but also outperforms the state-of-the-art scheme [93], especially for large-scale topologies or an increased number of time periods. The obtained results are analyzed on topologies from a real data set.

## A.2 Related Work

The comprehensive surveys in [100–102] provide detailed information about hybrid software-defined networks, their deployment solutions and optimization strategies. As suggested by [97], there are two primary lines of research regarding hybrid software-defined networks. First, solutions such as Fibbing [103] and SDNp [97] mostly focus on providing centralized control over distributed routing through introducing new network architectures. In such architectures, the central controller manages the traffic from legacy routers using special control messages. More details on the differences between the aforementioned architectures can be found in [97, 103]. Second, stacked hybrid models (more typical hybrid networks), involve adding high-priority rules to the forwarding tables of the SDN-enabled network devices (policy-based routing) by the SDN controller. Google's B4 [17] and Panopticon [104] exemplify the implementations of such a scheme. In stacked hybrid models, seeking an efficient migration strategy for an incremental SDN deployment is of great importance for the network operators. To this end, in [93,96,98,99,105,106], the researchers are mainly concerned with the modeling, analysis and algorithmic approaches to handle the problem, which is the focus of our work as well.

A genetic algorithm was proposed in [96] to find a migration sequence of legacy routers that minimizes the maximum link utilization (which is helpful to mitigate link congestion that may stem from link/node failures). They demonstrated the superiority of their proposed scheme by comparing it to greedy and static algorithms. Hong et al. [99] solved the aforementioned problem by proposing three greedy heuristic algorithms to upgrade the legacy routers with regard to their degree in the topology graph, their frequency of appearance in the K-shortest paths among all source and destination pairs, and their traffic volume. They evaluated these algorithms on real-world ISP and enterprise topologies and reported that there was no conspicuous difference in the results acquired by the algorithms. An incremental OpenFlow switch deployment for a hybrid software-defined network was proposed in [105]. The authors

provided a heuristic scheme to solve two problems, including the maximization of the total flow control ability of nodes in the network (for an SDN node, its flow control ability was defined as the total size of flows that passed through it) with a given upgrading budget constraint and minimizing the upgrading cost to obtain the best network control ability.

Caria et al. [98] investigated the problem of network migration to SDN in which the number of nodes that were upgraded per time period was restricted to the ratio between the total number of nodes and the number of migration periods. They showed that their proposed optimized migration schedule (that maximizes the number of alternative paths activated by SDN-enabled nodes) outperformed a random schedule for different network topologies. In another research [106] the cost constraints of SDN migration were taken into account as well and near-optimal greedy algorithms were proposed and compared with each other with regard to their computational complexities, running times, and network capacity gains. In the most recent work [93], the authors took a further step by analyzing the impact of the reduced equipment cost over time. In particular, they modeled the problem as the maximization of a set function with the bounded supermodular degree. They proposed a greedy algorithm which only guaranteed an approximation ratio for a special case of the problem where the upgrade costs were uniform (i.e., no reduction in the cost in each time period) and no comparison was made with the optimal solution for the general case. Moreover, the approach used in [93] is resilient to path failures [100].

While our assumptions for problem formulation are similar to [93], our proposed metaheuristic algorithm outperforms the one in [93]. Moreover, our performance evaluation is carried out on larger topologies and results are compared with the optimal solutions.

## A.3 System Model and Problem Formulation

The review of the related work on incremental migration to SDN in Section A.2 demonstrates two main optimization objectives that contribute to handling link/node failures more effectively in a hybrid software-defined network. The first one is minimizing the maximum link utilization (e.g., [96, 99]), which is a common traffic engineering goal among both legacy, hybrid, and full SDN-based networks to prevent link congestion that may result from node/link failures. The second one, which is of particular interest in hybrid software-defined networks, is maximizing the number

of alternative paths (e.g., [93, 98, 106]) that can be activated via SDN-enabled nodes. The number of dynamically selectable alternative paths increases as the number of SDN-enabled nodes rises (a flow that passes through more than one SDN-enabled nodes results in more dynamic routing options).

In this research, our optimization objective is to maximize the number of available alternative paths due to the following reasons. Maximizing the number of alternative paths allows for dynamic responses to failures or temporal link congestion since alternative paths can be enabled on demand, which further enhances the resilience of the hybrid software-defined networks with regard to the uncertainties of future network states [93]. Moreover, the traffic of backbone networks with a great number of aggregated flows is not expected to have a very bursty behavior within a time period. Hence, link capacity dimensioning can be performed at the beginning of each time period and the service provider can set a fair capacity headroom in terms of the maximum link utilization to ensure that the links are reasonably loaded in a time period. Finally, the link costs can be set inversely proportional to the capacity of the links without explicitly considering the link utilization [93, 98].

For each simple path (i.e., a path without loop) that is not the shortest path between a source and destination pair to be enabled as an alternative path, its *key nodes* [98] need to be identified. *Key nodes* are a set of nodes that must be upgraded to SDN-enabled nodes to activate an alternative path. The reason why such alternative paths cannot be used without enabling SDN features stems from the nature of destination-based conventional routing protocols that always leads to choosing the least cost path (i.e., the shortest path). It should be noted that in case of having more than one shortest paths between a source and destination pair one of them is considered as the primary one. To have a clearer picture, we give an example. As shown in Figure A.1 and Table A.1, there are four simple paths from node 3 to node 8 in the Abilene topology. Without migrating any node to SDN, all the flows from node 3 to node 8 are routed using the shortest path ($\{3, 6, 7, 8\}$). If node 6 is migrated to SDN, the path $\{3, 6, 4, 5, 8\}$ can also be employed to send the traffic from node 3 to node 8. This is due to the fact that node 6 can be configured by the SDN controller to forward the aforementioned traffic to node 4 instead of node 7. Similar explanations apply to the other alternative paths.

We assume $\boldsymbol{G}$=($\boldsymbol{N}$, $\boldsymbol{E}$) as an undirected weighted graph that represents the network topology ($\boldsymbol{N}$ denotes the set of routers and $\boldsymbol{E}$ indicates the set of links). The weights of the links can be the propagation latency (more preferable in WAN) or any

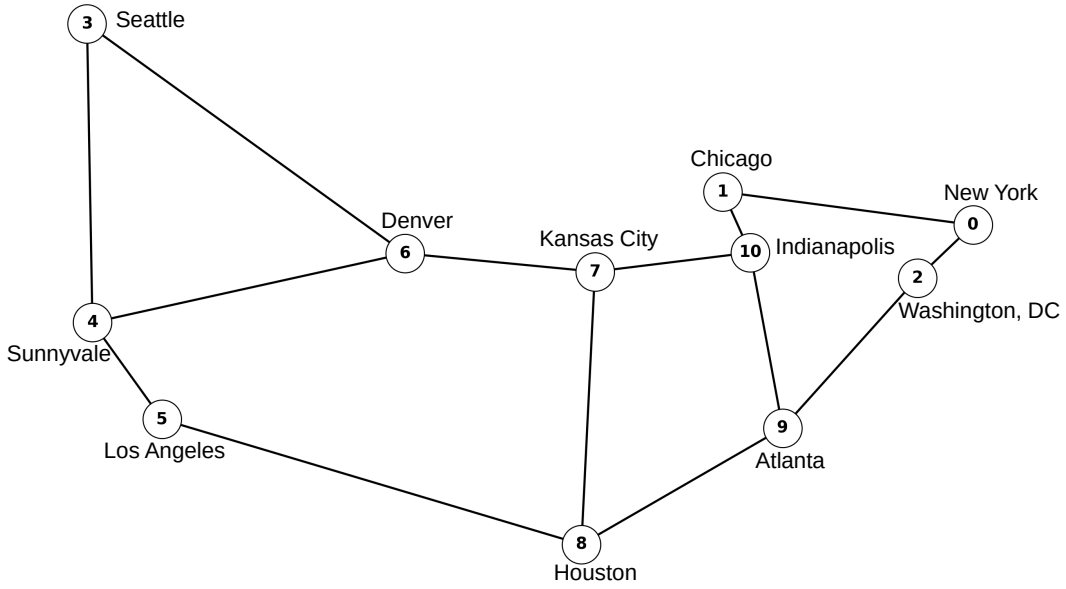



Figure A.1: The Abilene topology.

Table A.1: All the simple paths and their key nodes for the flows from Seattle (node 3) to Houston (node 8) considering the Abilene topology

| **Path** | **Path length (ms)** | **Key nodes** |
|---|---|---|
| $\{3, 6, 7, 8\}$ | 18.13 | $\{\}$ |
| $\{3, 6, 4, 5, 8\}$ | 29.71 | $\{6\}$ |
| $\{3, 4, 5, 8\}$ | 19.52 | $\{3\}$ |
| $\{3, 4, 6, 7, 8\}$ | 23.2 | $\{3, 4\}$ |

combination of desired metrics by the network operator such as OSPF weights. We denote the set of flows (commodities) by $\boldsymbol{F}$ and a flow $f$ is mainly identified by its source and destination. Also, the set of alternative paths for a flow $f$ is indicated by $\boldsymbol{P}_f$. We assume a sequence of $T_m$ periods (i.e., $\boldsymbol{T} = \{1, 2, 3, ..., T_m\}$) to exert the migration. The cost of upgrading a legacy router $n$ at period $t$ is denoted by $c_{n,t}$, where $c_{n,t} \leq c_{n,t'},\ t \geq t'$ holds. Moreover, the number of key nodes on an alternative path $p$ is indicated by $\omega_p$ whereas $\gamma_{n,p}$ indicates whether or not node $n$ is a key node on path $p$. We define two binary decision variables, namely $X_{n,t}$ and $Y_{p,f,t}$. While $X_{n,t}$ denotes whether node $n$ is upgraded to SDN in time period $t$ or not, $Y_{p,f,t}$ is set to 1 if an alternate path $p$ ($p \in \boldsymbol{P}_f$) is already available for flow $f$ in time period $t$ and 0 otherwise. Table A.2 summarizes the notation used in the problem formulation. While capturing all details, the resultant formulation is as follows.

Table A.2: Notation used in the problem formulation

| Symbol | Definition |
|---|---|
| $\boldsymbol{G}$ | Topology of an SD-WAN |
| $\boldsymbol{N}$ | Set of nodes (routers) |
| $\boldsymbol{E}$ | Set of links |
| $\boldsymbol{F}$ | Set of all flows |
| $f$ | A flow |
| $p$ | An alternative path |
| $\boldsymbol{P}_f$ | The set of alternative paths for flow $f$ |
| $T_m$ | The number of time periods |
| $\omega_p$ | The number of key nodes on an alternative path $p$ |
| $\gamma_{n,p}$ | 1 if node $n$ is a key node on path $p$, and 0 otherwise |
| $X_{n,t}$ | 1 if node $n$ is upgraded to SDN in time period $t$, and 0 otherwise |
| $Y_{p,f,t}$ | 1 if an alternate path $p$ is already available for flow $f$ in time period $t$, and 0 otherwise |
| $c_{n,t}$ | The cost of upgrading a legacy router $n$ at period $t$ |

$$\max \sum_{t\in\boldsymbol{T}}\sum_{f\in\boldsymbol{F}}\sum_{p\in\boldsymbol{P_f}} Y_{p,f,t} \tag{A.1}$$

subject to

- A key node can be upgraded to SDN at most once, i.e.,

$$\sum_{t\in\boldsymbol{T}} X_{n,t} \leq 1, \quad \forall n \in \boldsymbol{N}. \tag{A.2}$$

- The total upgrade cost should not exceed the available budget $B$ for the upgrade, i.e.,

$$\sum_{n\in\boldsymbol{N}}\sum_{t\in\boldsymbol{T}} c_{n,t}X_{n,t} \leq B. \tag{A.3}$$

- The availability of an alternative path in time period $t$ relies on the fact that all its key nodes have already been upgraded by that period (this constraint has been embedded in the objective function of [93]).

$$\omega_p\ Y_{p,f,t} \leq \sum_{t'\in\boldsymbol{T},\ t'\leq t}\ \sum_{n\in\boldsymbol{N}} \gamma_{n,p}X_{n,t'},\ \forall p \in \boldsymbol{P_f},\ \forall f \in \boldsymbol{F},\ \forall t \in \boldsymbol{T}. \tag{A.4}$$

- The integrality constraints are given by

$$X_{n,t},\ Y_{p,f,t} \in \{0,1\},\ \forall n \in \boldsymbol{N},\ \forall p \in \boldsymbol{P_f},\ \forall f \in \boldsymbol{F},\ \forall t \in \boldsymbol{T}. \tag{A.5}$$

## A.4 Proposed Algorithms

Due to the complexity and large solution space of the combinatorial optimization problem in Section A.3, especially for large values of $\boldsymbol{N}$ and $T_m$, in this section we propose two algorithms for solving the optimization problem.

### A.4.1 Defining a Rank Function

Before going through the details of our proposed algorithms, we first define a function $R$, which determines the rank (importance) of upgrading a key node $k$ in time period $t$ as follows.

$$R(k,t) = R_1(k,t) + R_2(k,t), \tag{A.6}$$

where

$$R_1(k,t) = \begin{cases} \frac{1}{t} & \text{if } k \text{ itself enables an alternative path.} \\ 0 & \text{otherwise.} \end{cases} \tag{A.7}$$

$$R_2(k,t) = \frac{\text{\# of paths having } k \text{ as their key node}}{c_{k,t}}. \tag{A.8}$$

$R_1$ reflects that if a key node itself can enable an alternative path (individual importance of a node), it increases the value of the objective function. However, the node importance decreases in later time periods. This is due to the fact that we prefer to benefit from the impact of SDN in earlier time periods by enabling alternative paths sooner. On the other hand, the upgrade operation is more beneficial to be done in later time periods since the upgrade costs decrease monotonically (lower cost results in a higher value for $R_2$). This results from the limited budget $B$ of the network operator, which is preferred to be utilized effectively to maximize the number of alternative paths. Note that the values of $R_1$ and $R_2$ are normalized (in $[0,1]$). It should be noted that migrating the key nodes based on a ranking scheme that only considers

**Algorithm 8** GReedy (GR)

1: $\boldsymbol{S} \leftarrow \varnothing$.
2: **while** *there exists an upgrade policy* $u$ *for a key node such that* $\boldsymbol{S} \cup \{u\}$ *is a feasible solution* **do**
3: Update the ranks of the potential upgrade policies.
4: Select the highest-ranked policy.
5: Update $\boldsymbol{S}$ and $B$.
6: **end while**

the number of alternative paths enabled by the key nodes, does not necessarily yield a good-quality solution [98]. In addition, a random upgrade sequence cannot fully take advantage of SDN capabilities [99].

### A.4.2 A Greedy Algorithm

*Algorithm* 8 summarizes the steps of our proposed greedy solution. Starting with an empty solution, while adding one of the potential upgrade policies results in a feasible solution, the algorithm proceeds according to steps 3–5. First, the ranks of potential upgrade policies are updated using function $R$. Then, in step 4, the upgrade policies are sorted in a decreasing order of their ranks, and the highest ranked upgrade policy that satisfies the budget constraint (i.e., its upgrade cost is less than or equal to the remaining total budget) is selected to be added to the solution. Finally, in step 5, $\boldsymbol{S}$ and budget $B$ are updated before choosing another upgrade policy.

To analyze the time complexity of *Algorithm* 8, it should be noted that the number of possible upgrade policies is upper bounded by $|\boldsymbol{N}| \times T_m$. Moreover, according to the constraint in (A.2), when a key node is chosen for upgrade, other policies involving the upgrade of the same node (in other time periods) are removed from the set of all potential upgrade policies. Therefore, the maximum number of iterations in the outer loop is $|\boldsymbol{N}|$ (results in $O(|\boldsymbol{N}|)$ complexity). Similarly, the time complexity of updating ranks is $O(|\boldsymbol{N}|)$ whereas the sort operation is performed in $O(|\boldsymbol{N}| \log |\boldsymbol{N}|)$ time. Finally, the overall time complexity of *Algorithm* 8 is $O(|\boldsymbol{N}|^2 \log(|\boldsymbol{N}|)$.

### A.4.3 A Constructive Metaheuristic Algorithm

Updating the ranks of the candidate solution components before selecting a new one, may not necessarily result in a good solution, especially for large problem instances. To enhance the constructive procedure of GR, we introduce a stochastic and construc-

tive metaheuristic algorithm. In contrast to exact methods, metaheuristic methods (as upper-level general methodologies) make it possible to tackle complex and large-size problem instances by generating high-quality solutions in a reasonable time for practical use. However, there is no guarantee to find optimal or bounded solutions. Moreover, approximation algorithms (which provide bounded solutions) are problem-dependent and thus, this limits their applicability. Additionally, some approximation-based solutions have a large gap with the global optimal solution, which causes them not to be tailored to many real-life applications. Therefore, metaheuristic algorithms have gained momentum and popularity in recent years [107].

Our proposed metaheuristic algorithm is based on ACO (Ant Colony Optimization) and we name it ACO-R (an ACO Rank-based algorithm). ACO has achieved great success in solving combinatorial optimization problems such as scheduling, routing, and assignment problems. The research in [108,109] exemplifies some of the recent applications of this approach in the context of communication networks. Inspired by real ants, the ACO-based algorithms rely on a set of artificial ants that build solutions in a randomized manner. The stochastic nature of ACO enables the artificial ants to construct a wide range of various solutions and thus explore a much larger number of solutions compared with greedy heuristics. The amounts of pheromones (denoted by $\tau$), which are deposited by each ant as a means of sharing their search experience with other ants, are regulated by a parameter called evaporation rate $\rho$ that affects the convergence speed of the algorithm. Pheromone trail evaporation is required to prevent a very fast convergence of the algorithm towards a sub-optimal solution. It provides a helpful form of forgetting, which favors the exploration of new regions of the search space. The probabilistic decisions are made as a function of the pheromone trails and the heuristic information (which is calculated using a heuristic function). The heuristic information helps the ants to move towards the elite solutions. Moreover, sharing the search experience among the ants can affect the solution construction in the future iterations of ACO [110].

ACO-R (shown in *Algorithm* 9) uses a heuristic function, $\eta$, that selects a solution component according to its rank (importance) using the rank function $R$ which was defined in Section A.4.1. The transition probability for choosing a solution component at each construction step (i.e., upgrading key node $k$ in time period $t$ by ant $a$) is as

**Algorithm 9** ACO-R

```
Initialization().
while i < CIters ∧ TIters < MIters do
    for j = 1 to Ants do
        while there exists an upgrade policy that results in a feasible solution do
            p ← random(0, 1).
            if p < Q_0 then
                next-policy = Exploitation().
            else
                next-policy = Exploration().
            end if
            Update the partial solution with the next-policy.
        end while
        Choose the best solution so far.
    end for
    TIters ← TIters + 1.
    if LastBestPolicy == BestPolicy then
        i ← i + 1.
    else
        i ← 0.
        LastBestPolicy ← BestPolicy.
    end if
    Update the pheromone trails (τ).
end while
```

follows

$$P_{kt}^{a} = \frac{\tau_{kt}^{\alpha}\eta_{kt}^{\beta}}{\sum_{kt \in \boldsymbol{S_c}} \tau_{kt}^{\alpha}\eta_{kt}^{\beta}}, \tag{A.9}$$

where $\mathbf{S}_c$ is the set of feasible solution components while $\alpha$ and $\beta$ indicate the relative impact of pheromone trails and the defined heuristic on the algorithm behavior in terms of exploration (unvisited regions of the search space) and exploitation (of accumulated search experience). The process of constructing solutions by ants is performed iteratively and the solution components that have been present in high-quality solutions will have a larger chance of being chosen.

The first step of *Algorithm* 9 includes the initialization of the ACO parameters and the matrix of pheromone trails ($\tau_{|\boldsymbol{N}|\times T_m}$) as well as the computation of $\eta$ (corresponding to the value of $R$) for all of the potential upgrade policies. The pheromone matrix

is initialized by the same value for all the pheromone trails and during the search it is updated to estimate the utility of an element $\tau_{kt}$ [107]. That is, each element of the $\tau$ matrix reflects the desirability of upgrading a key node $k$ in time period $t$. $MIters$, $TIters$, $Ants$ and $CIters$ denote the maximum number of iterations, the total number of iterations, the number of ants and the number of consecutive iterations that lead to the same value for the objective function, respectively. ACO-R ends when $i$ reaches $CIters$ or the maximum number of iterations is reached (step 2). Each ant constructs a solution by selecting a set of node upgrade policies using exploitation or exploration (steps 3–14). In particular, when exploitation is used, the previous search experience is taken into account. Thus, the next node upgrade policy which has the highest value of $\tau_{kt}^{\alpha}\eta_{kt}^{\beta}$ is selected. On the other hand, when the exploration is utilized, the next node upgrade policy is chosen based on the probability in (A.9) without considering the search experience of other ants. It should be noted that according to the constraint in (A.2), when a key node is chosen for upgrade, other policies involving the upgrade of the same node (in other time periods) are removed from the set of all potential upgrade policies. Moreover, at both of the exploitation and exploration phases, a key node is only selected if its upgrade cost is less than or equal to the remaining budget and the remaining budget is updated after each node upgrade. Finally, as the last step of the `while` loop, if the best solution found by all of the ants in the current iteration has already been found in previous iterations, counter $i$ associated with $CIters$ is incremented; Otherwise, it is reset.

## A.5 Performance Evaluation

In this section, we analyze the acquired results on the topologies from the ITZ data set with regard to the total budget $B$ and the number of time periods ($T_m$). Moreover, we analyze the performance of our proposed metaheuristic algorithm (ACO-R) with regard to the solution quality of its obtained solutions, its computational effort, and its robustness.

### A.5.1 Experiment Setup

The chosen topologies range from small scale (Abilene with 11 nodes and 14 edges) to medium scale (DFN with 51 nodes and 80 links) and large scale (TATA with 143 nodes and 181 links and Cogent with 180 nodes and 210 links). We considered

Table A.3: The parameters of ACO-R and their associated values

| **ACO-R Parameter** | **Value** |
|---|---|
| $\rho$ | 0.1 |
| $\alpha$ | 0.2 |
| $\beta$ | 2 |
| $\tau_{kt}$ (initial value) | 0.0001 |
| $Q_0$ | 0.5 |
| $MIters$ | 500 |
| $CIters$ | 100 |
| $Ants$ (number of ants) | 10 |

uniform initial upgrade costs ($\$100K$) and $C_r = 40\%$ (upgrade cost reduction rate). Also, as a pre-processing step, for each possible flow in a topology, two alternative paths (second and third shortest paths based on the propagation latency) were computed and their key nodes were identified. Table A.3 shows the parameter settings for ACO-R. The values of ACO parameters, including $\rho$, $\alpha$, and $\beta$, were set according to the guidelines in [110]. The same weight was assigned to exploration and exploitation ($Q_0 = 0.5$). ACO-R was run 30 times and the maximum value of the outcomes of these experiments was chosen (due to its randomized nature). As we mentioned earlier, the stopping criteria for each run of ACO-R is reaching the maximum number of iterations or when the solution quality does not change for a number of consecutive iterations. We implemented the proposed algorithms using Python and all the experiments were carried out on an Intel(R) Core i7-7660U CPU @2.50GHz and 8GB RAM with Windows 10 Pro (64-bit) installed. Gurobi optimization software (version 7.5.2) was used to obtain the optimal solution (i.e., OPT).

### A.5.2 Comparison with the Existing Work

We compared our proposed algorithms with the most recent work, SGR [93] and DEG [99]. SGR gives priority to a solution component with the maximum marginal value. It should be noted that unlike SGR, we assumed the same priority for all the flows although different priorities can be included in our objective function. DEG is a static algorithm based on the degree centrality of the key nodes and the assumption of having a single time period. The reasoning behind this heuristic algorithm is that the nodes with a higher number of connections are more likely to be passed by routing paths.

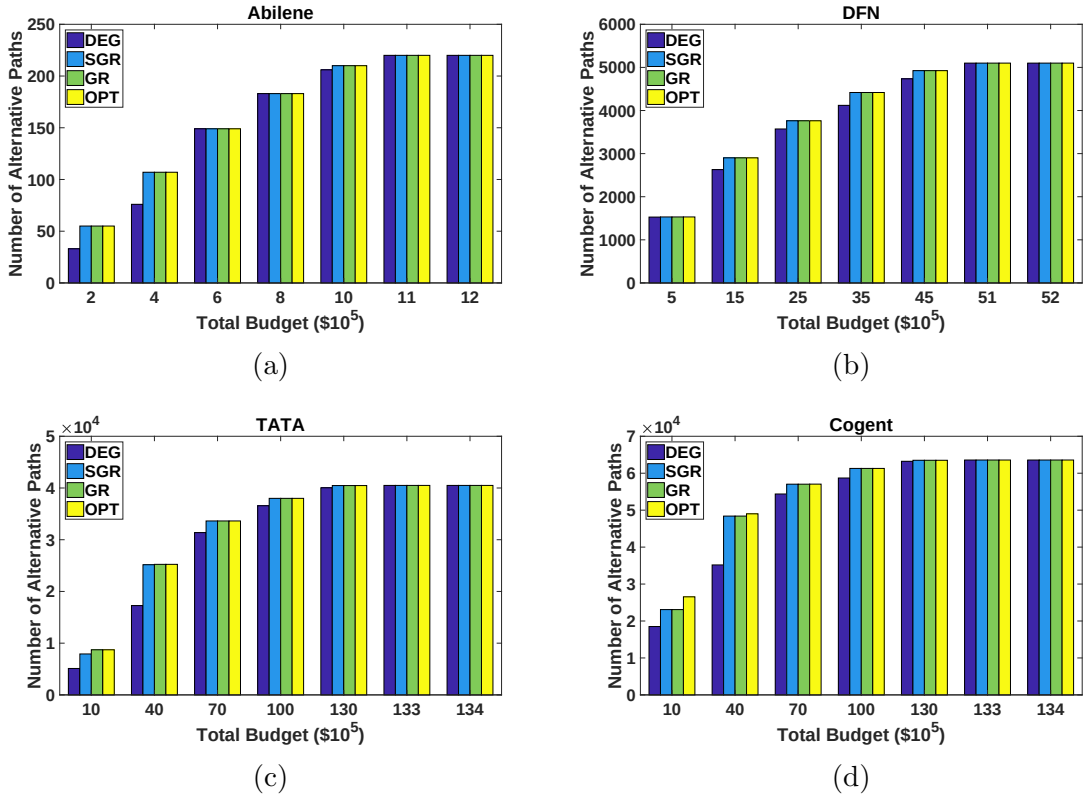



Figure A.2: The impact of increasing $B$ on the number of enabled alternative paths.

**The impact of increasing total budget**: As shown in Figure A.2a, Figure A.2b, Figure A.2c, and Figure A.2d, for $T_m = 1$, by increasing the total budget $B$, the number of enabled alternative paths rises since more key nodes can be upgraded. As we mentioned earlier, for each possible flow in a topology, two alternative paths are computed. Hence, the maximum number of alternative paths (for $\boldsymbol{T} = \{1\}$) is $(|\boldsymbol{N}| \times |\boldsymbol{N}| - |\boldsymbol{N}|) \times 2$ (e.g., for Abilene this value is 220). For the Abilene topology, after $B = \$1100K$, the number of enabled alternative paths remains constant since this is the saturation point (increasing the budget will not result in any gain since all the available paths have been already activated). Similar results are obtained for DFN ($B = \$5100K$), TATA, and Cogent ($B = \$13300K$, some of the flows have only one alternative path) as well, which are related to the size of the topology (i.e., the number of nodes) and its number of edges. Therefore, having a sufficiently large amount of budget (the budget constraint becomes more relaxed) leads to the same objective value for all the algorithms.

As illustrated in Figure A.2, DEG has a lower performance than the other schemes for all topologies due to its reliance on only the degree of the nodes to be upgraded.

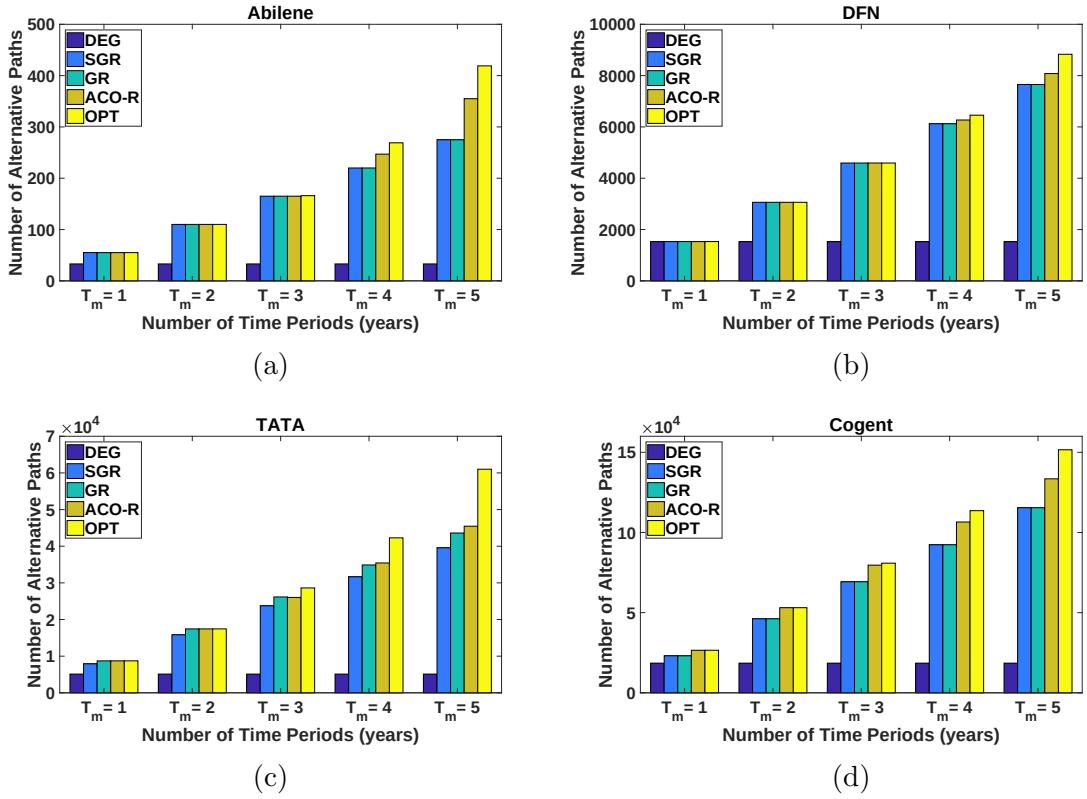



Figure A.3: The impact of increasing $T_m$ on the number of enabled alternative paths.

While both SGR and GR result in optimal solutions for small and medium size topologies (Abilene and DFN), this is not always the case for larger topologies. Considering TATA, for $B = \$1000K$ and $B = \$4000K$, GR achieves the optimal solution whereas SGR has a lower performance (although the maximum optimality gap between GR and SGR in this case is around 0.3% for $B = \$4000K$). It should be noted that since using our proposed greedy algorithm (GR) leads to acquiring the optimal solution (OPT) for most of the topologies when increasing the budget in a single time period, we did not run the ACO-R for these scenarios. Although for the largest topology, Cogent, both GR and SGR have the same performance in case of $B \in \{\$1000K, \$4000K, \$7000K\}$ that is not optimal, the optimality gap is very small ($< 1.3\%$) in case of $B = \$4000K$ and $B = \$7000K$. For $B = \$1000K$, the gap is around 13% and thus, we ran ACO-R which achieved the optimal solution.

**The impact of increasing the number of time periods**: For $T_m > 1$, the objective value is improved by postponing some of the upgrade operations to later time periods since the cost will be lower (as illustrated in Figure A.3). A practical value for $T_m$ is usually between 3 to 5 years [93]. The total allocated budgets to

Abilene and DFN are \$200K and \$500K, respectively while for TATA and Cogent the total budget is \$1M. This is due to the fact that, in practice, larger networks require a higher investment for the progressive upgrade to SDN. ACO-R outperforms GR, SGR, and DEG for $T_m > 3$. Furthermore, for $T_m \leq 3$ the outcome of ACO-R is in par with the one from SGR and GR for the small-scale and medium-scale topologies, i.e., Abilene and DFN, whereas it is superior to the results of SGR for the large-scale topologies (i.e., TATA and Cogent). This substantiates the fact that ACO-R is beneficial for finding a sub-optimal migration sequence for larger topologies or higher values of $T_m$ for which the problem becomes more complex. Regarding GR, in most cases it achieves the same performance as SGR; however, its performance is superior to SGR for TATA. It should be noted that it is not always possible to obtain the optimal solution for more complex scenarios and larger problem instances using optimization solvers such as Gurobi. In this case, ACO-R is a viable and efficient solution.

### A.5.3 The Analysis of ACO-R

The key performance indicators of metaheuristtic algorithms are categorized into three groups, namely *solution quality*, *computational effort*, and *robustness*. The quality of a solution obtained from a metaheuristic algorithm is usually determined by its gap to the global optimal solution, or a tight lower/upper bound of the solution (lower bound for a minimization problem and upper bound for a maximization problem), or the best-known solution (a reference solution). While using computation time as an indicator of the computational effort of a metaheuristic algorithm has the disadvantage of being dependent on computer system characteristics (such as CPU, memory, and operating system), other indicators such as the number of iterations to obtain the best solution are independent of the computer system. Considering robustness, there are a number of definitions. In general it is defined as the insensitivity of a metaheuristic algorithm against small deviations in its input instances or its parameters. A robust algorithm should perform well on different problem instances using the same parameters. Moreover, regarding the stochastic algorithms (such as ACO), robustness can be determined with regard to the average/deviation behavior of the algorithm over various runs on the same problem instance [107].

**Solution quality**: Regarding the quality of the solutions obtained from ACO-R, we utilize the gap with the optimal solution as well as the gap with the best known

Table A.4: Improved solution quality by ACO-R compared with SGR for different topologies

| **Topology** | $T_m = 1$ | $T_m = 2$ | $T_m = 3$ | $T_m = 4$ | $T_m = 5$ |
|---|---|---|---|---|---|
| Abilene | 0% | 0% | 0% | 12.27% | 29.09% |
| DFN | 0% | 0% | 0% | 2.38% | 5.56% |
| TATA | 9.4% | 10.16% | 9.6% | 11.9% | 14.8% |
| Cogent | 14.96% | 14.96% | 14.96% | 15.36% | 15.61% |

solution (acquired by SGR). This has been illustrated by Figure A.3 and more details about the gap between ACO-R and SGR are given in Table A.4. It is clear that ACO-R outperforms SGR when the topology size and/or the number of time periods increases.

**Computational effort**: We utilize the number of iterations to the best solutions as a performance indicator for the computational effort. Figure A.4 depicts the average number of iterations to converge to the best solution with regard to each topology and the number of time periods. As it can be seen, the average number of iterations increases not only when the number of time periods rises but also when the topology size is scaled up (in both cases the complexity of the problems rises that subsequently increases the average number of iterations to obtain the best solution).

**Robustness**: The average/standard deviation of the objective value is used as a measure for robustness. Figure A.5 illustrates the average and standard deviation of the objective value (i.e., the number of alternative paths) with regard to each time period for all topologies. As it can be seen, there is not too much deviation in the results. It should be noted that, as shown in Table A.3, the values of the parameters of ACO-R are fixed for the experiments carried out on all topologies. This also substantiates the robustness of ACO-R.

**The impact of increasing the evaporation rate of ACO-R**: It is worth mentioning that changing the evaporation rate ($\rho$) affects the solution quality and the computational effort of ACO-R. While it is suggested to follow the guidelines to set ACO parameters such as $\alpha$ and $\beta$, we can change the value of $\rho$ to decrease/increase the convergence speed of the algorithm. As shown in Table A.3, we chose a small value for $\rho$ to have high quality solutions. However, this comes with the cost of increased computational effort and a slower convergence speed. As an example, Figure A.6 illustrates the impact of increasing $\rho$ on the solution quality and computational effort in a run of ACO-R for $T_m = 5$ and TATA topology. It can be seen that a higher

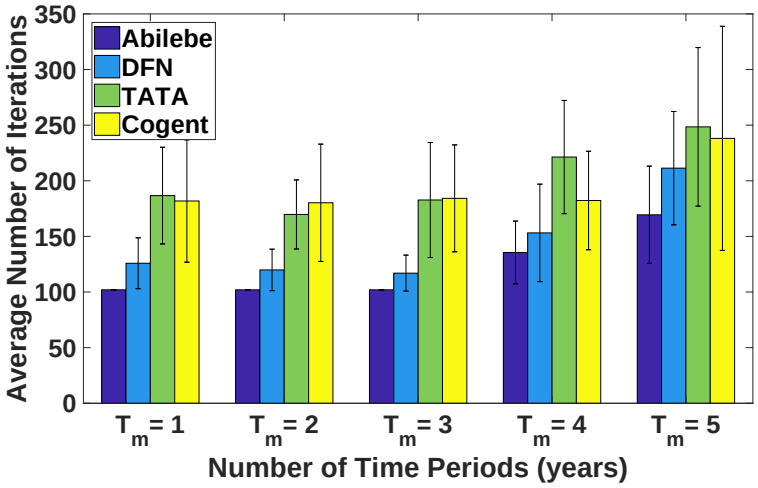



Figure A.4: The average number of ACO-R iterations for each time period and different topologies as a measure of computational effort.

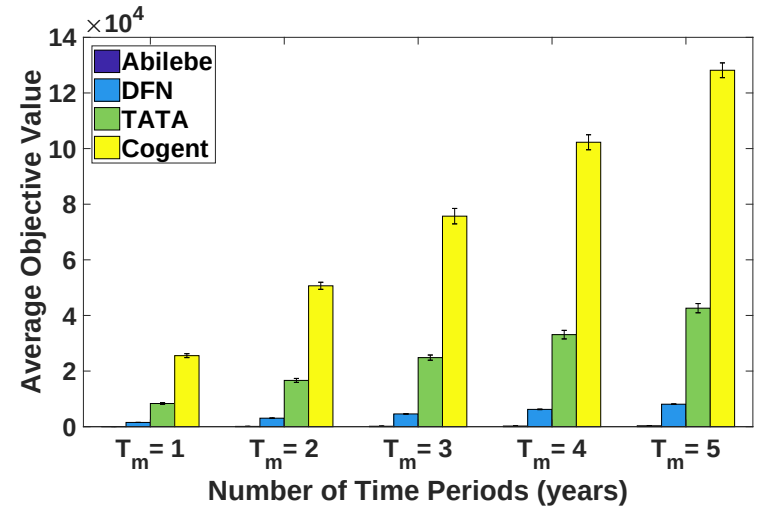



Figure A.5: The average objective value using ACO-R for each time period and different topologies as a measure of robustness (for 30 runs).

evaporation rate increases the convergence speed which subsequently reduces the solution quality (the objective value is lower for larger values of $\rho$) and computational effort (i.e., the number of iterations).

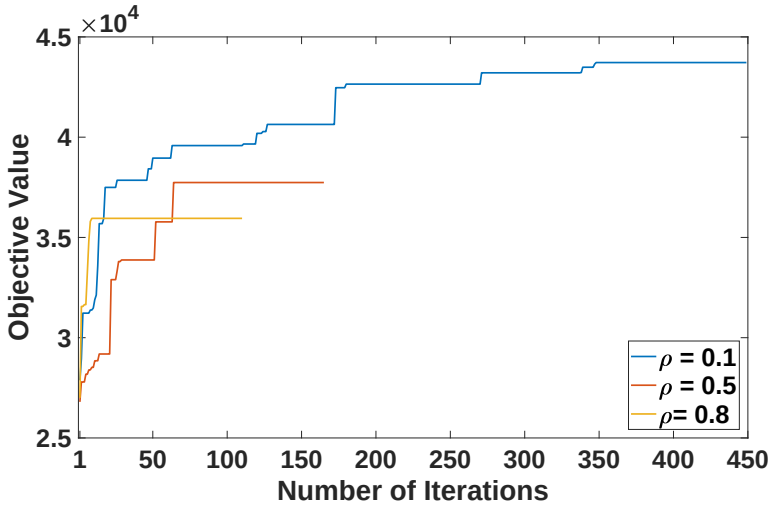



Figure A.6: The impact of increasing $\rho$ on the solution quality and computational effort.

## A.6 Conclusion

In this research work, we investigated the incremental migration to SDN such that it improves traffic engineering flexibility while taking into account the migration budget constraints. This subsequently can assist in providing certain responses to network dynamics such as node/link failures in the data plane. We proposed a greedy algorithm as well as a metaheuristic algorithm to solve the aforementioned NP-hard problem. Through extensive simulations, we evaluated the performance of the proposed schemes and showed that they outperform the state-of-the-art scheme, especially for large-scale and more complex problem instances. Moreover, an analysis of the metaheuristic algorithm was provided that substantiated its effectiveness.

# Appendix B

# Selected Publications

The following list of publications shows the outcome of my research and collaboration, as the main author, with my supervisor, lab mates, and lab visitors during my PhD program at University of Victoria. Considering the list, while the first paper reflects the research carried out in Chapter 3, the second and third ones represent the research conducted in Chapter 2 and Appendix A, respectively. The fourth paper is focused on failure recovery in the data plane of software-defined wireless mesh networks. Moreover, the fifth paper addresses a similar problem to Chapter 2; however, it proposes a different problem formulation and modeling of the problem. Finally, the sixth paper provides a centralized scheme for (re)routing the flows in wireless networks as well as designing a metaheuristic algorithm to solve the associated optimization problem (similar to Appendix A).

1. **M. Tanha**, D. Sajjadi, R. Ruby, and J. Pan, "Resilient switch reassignment and incremental controller placement for software defined WANs," *submitted to IEEE Trans. Network Serv. Manage, Sep 2018, currently under revision by authors.*

2. **M. Tanha**, D. Sajjadi, R. Ruby, and J. Pan, "Capacity-aware and delay-guaranteed resilient controller placement for software defined WANs," *IEEE Trans. Network Serv. Manage*, vol. 22, no. 3, pp. 438–411, 2018.

3. **M. Tanha**, D. Sajjadi, R. Ruby, and J. Pan, "Traffic engineering enhancement by progressive migration to SDN," *IEEE Commun. Lett.*, vol. 22, no. 3, pp. 438–441, 2018.

4. **M. Tanha**, D. Sajjadi, and J. Pan, "Demystifying failure recovery for software defined wireless mesh networks," in *Proc. IEEE NetSoft (PVE-SDN workshop)*, 2018.

5. **M. Tanha**, D. Sajjadi, and J. Pan, "Enduring node failures through resilient controller placement for software defined networks," in *Proc. IEEE GLOBECOM*, 2016.

6. **M. Tanha**, D. Sajjadi, F. Tong, and J. Pan, "Disaster management and response for modern cellular networks using flow-based multi-hop device-to-device communications, in *Proc. IEEE VTC2016-Fall*, 2016.

# Bibliography

[1] Centre for Research on the Epidemiology of Disasters, "Annual Disaster Statistical Review 2017," https://www.emdat.be/publications.

[2] M. F. Habib, M. Tornatore, F. Dikbiyik, and B. Mukherjee, "Disaster survivability in optical communication networks," *Comput. Commun.*, vol. 36, no. 6, pp. 630–644, 2013.

[3] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Comput. Networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[4] T. Sakano, Z. M. Fadlullah, T. Ngo, H. Nishiyama, M. Nakazawa, F. Adachi, N. Kato, A. Takahara, T. Kumagai, H. Kasahara *et al.*, "Disaster-resilient networking: A new vision based on movable and deployable resource units," *IEEE Network*, vol. 27, no. 4, pp. 40–46, 2013.

[5] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, "Resilience support in software-defined networking: A survey," *Comput. Networks*, vol. 92, pp. 189–207, 2015.

[6] B. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 2014.

[7] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Comput. Networks*, vol. 71, pp. 1–30, 2014.

[8] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 493–512, 2014.

[9] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 70–75, 2014.

[10] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 473–478, 2012.

[11] S. N. Rizvi, D. Raumer, F. Wohlfart, and G. Carle, "Towards carrier grade SDNs," *Comput. Networks*, vol. 92, pp. 218–226, 2015.

[12] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 333–354, 2017.

[13] M. F. Bari, "Resource Orchestration in Softwarized Networks," Ph.D. dissertation, University of Waterloo, 2018.

[14] Y. Jiménez, C. Cervelló-Pastor, and A. J. García, "On the controller placement for designing a distributed SDN control layer," in *Proc. IFIP NETWORKING*, 2014, pp. 1–9.

[15] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "ElastiCon; An elastic distributed SDN controller," in *Proc. ACM/IEEE ANCS*, 2014, pp. 17–27.

[16] J. English, "2016 SDN trends: The year of the software-defined WAN," https://goo.gl/a2i9CY, 2016.

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 3–14.

[18] R. Ahmed and R. Boutaba, "Design considerations for managing wide area software defined networks," *IEEE Commun. Mag.*, vol. 52, no. 7, pp. 116–123, 2014.

[19] D. Tipper, "Resilient network design: Challenges and future directions," *Telecommunication Systems*, vol. 56, no. 1, pp. 5–16, 2014.

[20] ONF, "OpenFlow Switch Specification-version 1.5.1," https://goo.gl/jE2JTW, 2015.

[21] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," *J. Netw. Comput. Appl.*, 2017.

[22] A. K. Singh and S. Srivastava, "A survey and classification of controller placement problem in SDN," *Int. J. Network Manage.*, vol. 28, no. 3, 2018.

[23] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The controller placement problem in software defined networking: A survey," *IEEE Network*, vol. 31, no. 5, pp. 21–27, 2017.

[24] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 19, no. 1, pp. 30–33, 2015.

[25] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1339–1342, 2014.

[26] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for optimal placement of SDN controllers," in *Proc. IEEE ICC*, 2016, pp. 1–6.

[27] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A K-means-based network partition algorithm for controller placement in software defined network," in *Proc. IEEE ICC*, 2016, pp. 1–6.

[28] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 4–17, 2015.

[29] M. Tanha, D. Sajjadi, and J. Pan, "Enduring node failures through resilient controller placement for software defined networks," in *Proc. IEEE GLOBECOM*, 2016, pp. 1–7.

[30] B. P. R. Killi and S. V. Rao, "Capacitated next controller placement in software defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 514–527, 2017.

[31] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The SDN controller placement problem for WAN," in *Proc. IEEE ICC*, 2014, pp. 220–224.

[32] L. Han, Z. Li, W. Liu, K. Dai, and W. Qu, "Minimum control latency of SDN controller placement," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 2175–2180.

[33] H. Kuang, Y. Qiu, R. Li, and X. Liu, "A hierarchical k-means algorithm for controller placement in SDN-based WAN architecture," in *Proc. IEEE ICMTMA*, 2018, pp. 263–267.

[34] H. Bo, W. Youke, W. Chuan'an, and W. Ying, "The controller placement problem for software-defined networks," in *Proc. IEEE ICCC*, 2016, pp. 2435–2439.

[35] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proc. IEEE CNSM*, 2013, pp. 18–25.

[36] H. K. Rath, V. Revoori, S. M. Nadaf, and A. Simha, "Optimal controller placement in Software Defined Networks (SDN) using a non-zero-sum game," in *Proc. IEEE WoWMoM*, 2014, pp. 1–6.

[37] N. Perrot and T. Reynaud, "Optimal placement of controllers in a resilient SDN architecture," in *Proc. DRCN*, 2016, pp. 145–151.

[38] Q. Zhong, Y. Wang, W. Li, and X. Qiu, "A min-cover based controller placement approach to build reliable control network in SDN," in *Proc. IEEE/IFIP NOMS*, 2016, pp. 481–487.

[39] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-scale dynamic controller placement," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 1, pp. 63–76, 2017.

[40] T. Y. Cheng, M. Wang, and X. Jia, "QoS-guaranteed controller placement in SDN," in *Proc. IEEE GLOBECOM*, 2015, pp. 1–6.

[41] A. Sallahi and M. St-Hilaire, "Expansion model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 21, no. 2, pp. 274–277, 2017.

[42] T. Zhang, P. Giaccone, A. Bianco, and S. De Domenico, "The role of the inter-controller consensus in the placement of distributed SDN controllers," *Comput. Commun.*, vol. 113, pp. 1–13, 2017.

[43] L. Zhu, R. Chai, and Q. Chen, "Control plane delay minimization based SDN controller placement scheme," in *Proc. IEEE WCSP*, 2017, pp. 1–6.

[44] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Hot-ICE*, 2012, pp. 1–6.

[45] L. F. Muller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos, "Survivor: An enhanced controller placement strategy for improving SDN survivability," in *Proc. IEEE GLOBECOM*, 2014, pp. 1909–1915.

[46] B. P. R. Killi and S. V. Rao, "Optimal model for failure foresight capacitated controller placement in software-defined networks," *IEEE Commun. Lett.*, vol. 20, no. 6, pp. 1108–1111, 2016.

[47] T. Yuan, X. Huang, M. Ma, and J. Yuan, "Balance-based SDN controller placement and assignment with minimum weight matching," in *Proc. IEEE ICC*, 2018, pp. 1–6.

[48] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," *Comput. Networks*, vol. 112, pp. 24–35, 2017.

[49] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient SDN control plane," in *Proc. RNDM*, 2016, pp. 253–259.

[50] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for software-defined networks," *China Commun.*, vol. 11, no. 2, pp. 38–54, 2014.

[51] F. J. Ros and P. M. Ruiz, "On reliable controller placements in software-defined networks," *Comput. Commun.*, vol. 77, pp. 41–51, 2016.

[52] A. Alshamrani, S. Guha, S. Pisharody, A. Chowdhary, and D. Huang, "Fault tolerant controller placement in distributed SDN environments," in *Proc. IEEE ICC*, 2018, pp. 1–7.

[53] P. Talhar and A. P. Bhagat, "An adaptive approach for controller placement problem in software defined networks," in *Proc. IEEE RICE*, 2018, pp. 1–11.

[54] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 30–36, 2016.

[55] M. Abu-Lebdeh, D. Naboulsi, R. Glitho, and C. W. Tchouati, "On the placement of VNF managers in large-scale and distributed NFV systems," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 875–889, 2017.

[56] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Trans. on Cloud Comput.*, vol. 2, no. 4, pp. 436–447, 2014.

[57] N. Beheshti and Y. Zhang, "Fast failover for control traffic in software-defined networks," in *Proc. IEEE GLOBECOM*, 2012, pp. 2665–2670.

[58] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *Proc. IEEE GLOBECOM*, 2011, pp. 1–6.

[59] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *Proc. IFIP/IEEE IM*, 2013, pp. 672–675.

[60] M. Guo and P. Bhattacharya, "Controller placement for improving resilience of software-defined networks," in *Proc. ICNDC*, 2013, pp. 23–27.

[61] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. ITC*, 2013, pp. 1–9.

[62] D. Hock, S. Gebert, M. Hartmann, T. Zinner, and P. Tran-Gia, "POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. IEEE/IFIP NOMS*, 2014, pp. 1–2.

[63] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia, "POCO-PLC: Enabling dynamic pareto-optimal resilient controller placement in SDN networks," in *Proc. IEEE INFOCOM WKSHPS*, 2014, pp. 115–116.

[64] B. Behsaz, M. R. Salavatipour, and Z. Svitkina, "New approximation algorithms for the unsplittable capacitated facility location problem," *Algorithmica*, vol. 75, no. 1, pp. 53–83, 2016.

[65] G. P. McCormick, "Computability of global solutions to factorable nonconvex programs: Part I—convex underestimating problems," *Math. Program.*, vol. 10, no. 1, pp. 147–175, 1976.

[66] M. C. Golumbic and I. B.-A. Hartman, *Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications*. Springer Science & Business Media, 2005.

[67] V. Vassilevska, "Efficient algorithms for clique problems," *Information Processing Letters*, vol. 109, no. 4, pp. 254–257, 2009.

[68] J. W. Moon and L. Moser, "On cliques in graphs," *Isr. J. of Math.*, vol. 3, no. 1, pp. 23–28, 1965.

[69] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[70] B. Rosgen and L. Stewart, "Complexity results on graphs with few cliques," *Discrete Math. Theoret. Comput. Sci.*, vol. 9, no. 1, 2007.

[71] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, "A new algorithm for generating all the maximal independent sets," *SIAM J. Comput.*, vol. 6, no. 3, pp. 505–517, 1977.

[72] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *ACM J. Exp. Algorithmics*, vol. 18, pp. 3–1, 2013.

[73] S. S. Skiena, *The Algorithm Design Manual*. Springer-Verlag London, 2008.

[74] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, 2011.

[75] Y. Zhao, L. Iannone, and M. Riguidel, "On the performance of SDN controllers: A reality check," in *Proc. IEEE NFV-SDN*, 2015, pp. 79–85.

[76] S. Mallon, V. Gramoli, and G. Jourjon, "Are today's SDN controllers ready for primetime?" in *Proc. IEEE LCN*, 2016, pp. 325–332.

[77] "Gurobi Optimizer," http://www.gurobi.com/.

[78] Ericsson, "Wide Area Network Traffic Engineering: Meeting the Challenges of the Distributed Cloud," https://goo.gl/mhWjpd, 2014.

[79] Y. Cui, Z. Yang, S. Xiao, X. Wang, and S. Yan, "Traffic-aware virtual machine migration in topology-adaptive DCN," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3427–3440, 2017.

[80] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, 2017.

[81] X. Huang, S. Bian, Z. Shao, and H. Xu, "Dynamic switch-controller association and control devolution for sdn systems," in *IEEE ICC*, 2017, pp. 1–6.

[82] N. Mouawad, R. Naja, and S. Tohme, "Optimal and dynamic SDN controller placement," in *Proc. IEEE ICCA*, 2018, pp. 1–9.

[83] S. Hegde, R. Ajayghosh, S. G. Koolagudi, and S. Bhattacharya, "Dynamic controller placement in edge-core software defined networks," in *IEEE TENCON*, 2017, pp. 3153–3158.

[84] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge university press, 2011.

[85] J. Y.-T. Leung and C.-L. Li, "Scheduling with processing set restrictions: A survey," *Int. J. Prod. Econ.*, vol. 116, no. 2, pp. 251–262, 2008.

[86] J. Y.-T. Leungyy and C.-L. Li, "Scheduling with processing set restrictions: A literature update," *Int. J. Prod. Econ.*, vol. 175, pp. 1–11, 2016.

[87] D. Recalde, C. Rutten, P. Schuurman, and T. Vredeveld, "Local search performance guarantees for restricted related parallel machine scheduling," in *Latin American Symposium on Theoretical Informatics*. Springer, 2010, pp. 108–119.

[88] C. Rutten, D. Recalde, P. Schuurman, and T. Vredeveld, "Performance guarantees of jump neighborhoods on restricted related parallel machines," *Operations Research Letters*, vol. 40, no. 4, pp. 287–291, 2012.

[89] J. K. Lenstra, D. B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Math. Program.*, vol. 46, no. 1-3, pp. 259–271, 1990.

[90] W. Wong, A. Thakur, and S.-H. G. Chan, "An approximation algorithm for AP association under user migration cost constraint," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[91] M. Cello, Y. Xu, A. Walid, G. Wilfong, H. J. Chao, and M. Marchese, "Balcon: A distributed elastic SDN control via efficient switch migration," in *IEEE IC2E*, 2017, pp. 40–50.

[92] A. Basta, A. Blenk, H. B. Hassine, and W. Kellerer, "Towards a dynamic SDN virtualization layer: Control path migration protocol," in *IEEE CNSM*, 2015, pp. 354–359.

[93] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, "One step at a time: Optimizing SDN upgrades in ISP networks," in *Proc. IEEE INFOCOM*, 2017.

[94] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2015–2020," https://bit.ly/2DGxZZr, 2016.

[95] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[96] Y. Guo, Z. Wang, X. Yin, X. Shi, J. Wu, and H. Zhang, "Incremental deployment for traffic engineering in hybrid SDN network," in *Proc. IEEE IPCCC*, 2015, pp. 1–8.

[97] M. Cariay, A. Jukan, and M. Hoffmann, "SDN partitioning: A centralized control plane for distributed routing protocols," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 381–393, 2016.

[98] M. Caria, A. Jukan, and M. Hoffmann, "A performance study of network migration to SDN-enabled traffic engineering," in *Proc. IEEE GLOBECOM*, 2013, pp. 1391–1396.

[99] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao, "Incremental deployment of SDN in hybrid enterprise and ISP networks," in *Proc. ACM SOSR*, 2016, pp. 1–7.

[100] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN networks: A survey of existing approaches," *IEEE Commun. Surveys Tuts.*, 2018, early access.

[101] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, and S. Hu, "A survey of deployment solutions and optimization strategies for hybrid SDN networks," *IEEE Commun. Surveys Tuts.*, 2018, early access.

[102] Sandhya, Y. Sinha, and K. Haribabu, "A survey: Hybrid SDN." *J. Netw. Comput. Appl.*, vol. 100, pp. 35–55, 2017.

[103] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," *ACM SIGCOMM CCR*, vol. 45, no. 4, pp. 43–56, 2015.

[104] D. Levin, M. Canini, S. Schmid, F. Schaffert, A. Feldmann *et al.*, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks." in *Proc. USENIX ATC*, 2014, pp. 333–345.

[105] X. Jia, Y. Jiang, and Z. Guo, "Incremental switch deployment for hybrid software-defined networks," in *Proc. IEEE LCN*, 2016, pp. 571–574.

[106] T. Das, M. Caria, A. Jukan, and M. Hoffmann, "Insights on SDN migration trajectory," in *Proc. IEEE ICC*, 2015, pp. 5348–5353.

[107] E.-G. Talbi, *Metaheuristics: From Design to Implementation.* John Wiley & Sons, 2009, vol. 74.

[108] A. Amokrane, R. Langar, R. Boutaba, and G. Pujolle, "Online flow-based energy efficient management in wireless mesh networks," in *Proc. IEEE GLOBECOM*, 2013, pp. 329–335.

[109] D. Sajjadi, M. Tanha, and J. Pan, "Meta-heuristic solution for dynamic association control in virtualized multi-rate WLANs," in *Proc. IEEE LCN*, 2016, pp. 253–261.

[110] M. Dorigo and T. Stützle, "Ant Colony Optimization: Overview and Recent Advances," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Springer, 2010, pp. 227–263.