Stateless Orchestrator Synchronization In Multi-Layer Multi-Domain Networks

by

Alireza Tirehkar

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN TELECOMMUNICATIONS NETWORK
M.A.Sc.

MONTREAL, APRIL 28, 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Kim Khoa Nguyen, Thesis supervisor
Department of Electrical Engineering, École de technologie supérieure

Mr. Mohamed Cheriet, Thesis Co-Supervisor
Department of Systems Engineering, École de technologie supérieure

Mr. Lokman Sboui, Chair, Board of Examiners
Department of Systems Engineering, École de technologie supérieure

Mrs. Christine Tremblay, Member of the Jury
Department of Electrical Engineering, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON MAY 18, 2023

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude and appreciation to my supervisor, Professor Kim Khoa Nguyen for his continued support and assistance throughout my research. I would also like to extend my appreciation to my co-supervisor, Professor Mohamed Cheriet for his contributions to this thesis.

I would like to express my appreciation to my friends who have been a constant source of support and encouragement throughout my thesis journey.

I would like to express my heartfelt gratitude to my parents for their unwavering support. I'm also thankful to my brothers, their constant encouragement and belief in me have kept me motivated. Finally, I want to express my deepest appreciation to my wife, Diana, for her incredible support and understanding.

Synchronisation de l'orchestrateur sans état dans les réseaux multicouches multidomaines

Alireza Tirehkar

RÉSUMÉ

Dans un réseau multicouche (MLN), un orchestrateur peut être déployé pour coordonner plusieurs couches, telles que IP, OTN et DWDM. Chaque couche est un domaine administratif. Précision de la mise à jour de la vue de l'orchestrateur est un défi pour le MLN. Dans en raison de la structure hiérarchique des MLN, une seule panne dans une couche sous-jacente peut se propager aux couches supérieures, générant ainsi de nombreuses alarmes dans ces couches. Ces alarmes peuvent envoyer des mises à jour informations incorrectes de concernant la cause principale de l'échec source de la panne, qui aboutit à des décisions incorrectes une mauvaise décision de l'orchestrateur pour le routage et l'allocation des ressources.

Ainsi, l'ordre de mise à jour par d'envoie des mises à jour de différentes couches est cruciale à l'orchestrateur est très important dans MLN pour éviter toute confusion de l'orchestrateur puisque la relation. Déterminer l'ordre optimal est difficile, en raison de la cartographie flexible n'est pas facile à calculer. Des liens entre les différentes couches est complexe, et le temps de propagation des pannes des couches sous-jacentes aux couches supérieures. Dans ce mémoire, nous proposons une méthode pour mettre à jour l'orchestrateur afin de, , qui permet de garantir que la cause racine source de panne est signalée correctement, en tenant compte de la dépendance entre les différentes couches et du temps de propagation des pannes. En particulier, nous optimisons la fréquence d'envoi des messages de mise à jour des couches à l'orchestrateur. Notre méthode peut être implémentée dans l'orchestrateur pour demander périodiquement des mises à jour aux contrôleurs selon une fréquence optimisée.

Nous formulons un problème d'optimisation non linéaire de nombre entier pour la mise à jour de l'orchestrateur, puis proposons deux algorithmes pour approximer la probabilité d'échec optimale pour la mise à jour de l'orchestrateur solution optimale. Les résultats de simulation expérimentaux montrent que notre algorithme MLNOU proposé peut obtenir un quasi-optimal qui se rapproche en moyenne de 11.3 % de différence par rapport à la solution optimale, ce qui démontre l'efficacité de notre algorithme et surpasse clairement les solutions comparatives.

Mots-clés: Orchestration, réseau multicouche multidomaine, propagation des pannes, sans état, avec état

Stateless Orchestrator Synchronization In Multi-Layer Multi-Domain Networks

Alireza Tirehkar

ABSTRACT

In a Multi-Layer Network (MLN), an orchestrator can be deployed to coordinate multiple layers, such as IP, OTN, and DWDM. Each layer is an administrative domain. Accurately updating the orchestrator's network view is challenging in MLNs. Due to the hierarchical structure of MLNs, a single failure in an underlying layer can propagate to the upper layers, hence generating many alarms in each of these layers. These alarms may send incorrect updates of the root cause of the failure, which results in incorrect decisions of the orchestrator for routing and resource allocation.

Therefore, setting up the order of updating by different layers is crucial in MLN to avoid confusion in the orchestrator's network view. This task is challenging, due to the flexible mapping of links between different layers, and also to the failure propagation time from the underlying layers to the upper layers. In this thesis, we propose a method to update the orchestrator to ensure that the root cause is reported correctly, taking into account the dependency among different layers and failure propagation time. In particular, we optimize the frequency of sending update messages from layers to the orchestrator. Our proposed method can be implemented in the orchestrator to request updates from controllers periodically at an optimized rate.

We formulate an integer nonlinear optimization problem for updating the orchestrator and then propose two algorithms to approximate the optimal failure probability for updating the orchestrator solution. Simulation experimental results show that our proposed MLNOU algorithm can obtain a near-optimal which approximates on average 11.3% different from the optimal solution, which demonstrates the significant effect of our algorithm on the orchestrator's performance and clearly outperforms the baselines.

Keywords: Orchestration, multi-layer multi-domain network, failure propagation, stateless, stateful

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

MLN	Multi Layer Network
IP	Internet Protocol
MPLS	Multi-Protocol Label Switching
TS	Topology Server
DWDM	Dense Wavelength Division Multiplexing
OTN	Optical Transport Network
ONOS	Open Network Operating System
NETCONF	Network Configuration Protocol
YANG	Yet Another New Language
MLNOU	Multi-Layer Network Orchestrator Updating
MLN-SG	Multi-Layer Network Stochastic Greedy
MCK	Multiple-Choice Knapsack
RM	Resource Management

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

s	second
Kbps	kilobits per second
Mbps	megabits per second

INTRODUCTION

Context and motivations

In general, Multi-layer networks (MLNs) consist of two or three layers, where the lower layers (layer 0 and layer 1) are optical networks, and the upper layer is Internet Protocol/Multi-Protocol Label Switching (IP/MPLS) network [Alzahrani & Katib (2018)]. This integrated architecture benefits from IP traffic engineering and the high capacity of optical networks. In this thesis, each layer is a domain, and each domain is provided by a different vendor with a different controller. The fundamental approach to take advantage of both MLN technologies and multi-domain network features is network orchestration [Koulougli, Nguyen & Cheriet (2020)].

An orchestrator is required to coordinate different layers and different domains through the SDN paradigm [Szyrkowiec *et al.* (2018)]. The Topology Server (TS) [Muñoz *et al.* (2015b)] gathers the domain topology of each SDN domain controller, and the orchestrator maintains the global view of the network from this gathered information [Gossels, Choudhury & Rexford (2019)]. In addition, the orchestrator manages the network resources by calling services provided by the controllers in each layer [Mirkhanzadeh *et al.* (2018)]. For example, the orchestrator can use gathered information to provision end-to-end network services through multiple layers and multiple technologies and make optimized decisions [Casellas, Martínez, Vilalta & Muñoz (2018)].

Fig. 0.1 shows an MLN with three layers. Layer 0 is the Dense Wavelength Division Multiplexing (DWDM) network, layer 1 is the Optical Transport Network (OTN) network, and layer 2 is the IP network.

Except for the lowest layer, each of the other layers is composed of two different parts, a dependent and an independent part. The independent part contains physical links and the dependent part contains virtual links. By a given mapping function, virtual links are mapped to

a physical link or set of physical links in the underlying layers which use different technologies [Manzanares-Lopez, Muñoz-Gea, Malgosa-Sanahuja & Flores-de la Cruz (2019)]. For example, in Fig. 0.2(a), the dependent part of the IP layer includes two virtual links, A-C and A-D, which are mapped respectively to a physical link I-L and two physical links I-J-K in the DWDM layer. c_0, c_1, and c_2 are message costs from DWDM, OTN, and IP controller to the orchestrator, and B is the network's available bandwidth.

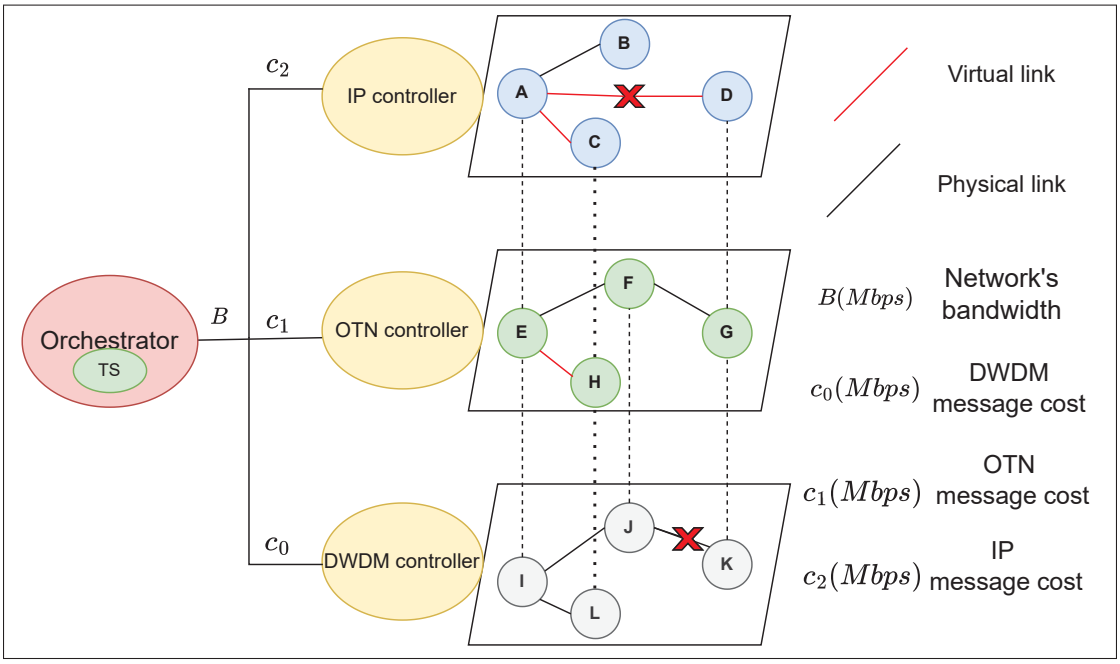
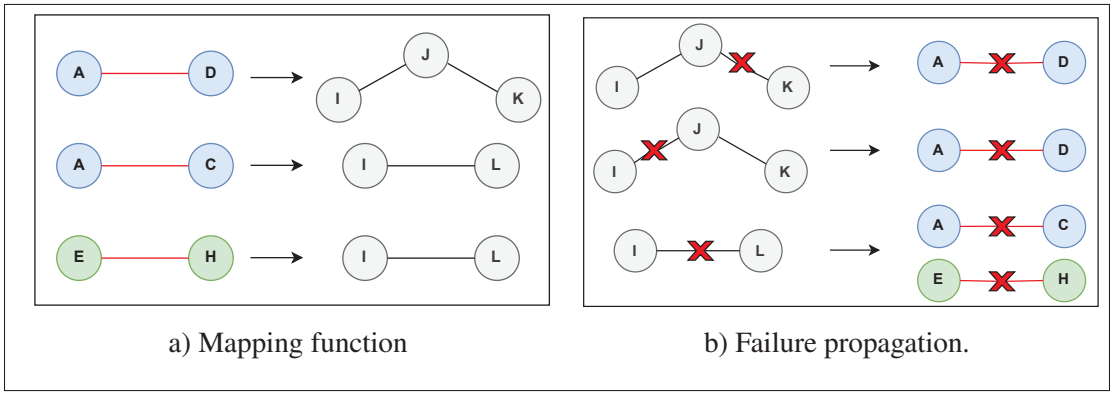


Figure 0.1 Hierarchical multi-layer multi-domain network with three layers



a) Mapping function b) Failure propagation.

Figure 0.2 Failure propagation and mapping between the layers

Due to the hierarchical structure of MLNs, one failure in the underlying layers may cause multiple failures in the dependent parts of the upper layers [Zhang, Wang, Jin & Song (2021a)]. Fig. 0.2(b) shows how single link failure in the underlying layers can propagate to the links in the upper layers. For instance, if physical link I-L in the DWDM layer fails, both virtual links A-C in the IP layer and E-H in the OTN layer will be failed. It is worth noting that both node and link failures can happen in the physical layer, however, based on the collected data of network operators, the probability of link failure is much higher than node failure [Gill, Jain & Nagappan (2011)]. Therefore, in this thesis, we focus on single link failure in the underlying layers and how it propagates to the upper layers.

The controllers update the orchestrator with the current state of their layer (e.g., available links) by sending messages to the TS. Several protocols can be used to coordinate the controllers and the TS. For instance, Open Network Operating System (ONOS) controller uses the Raft protocol for coordination among controllers [Zhu *et al.* (2020)]. In [Mayoral, Vilalta, Muñoz, Casellas & Martínez (2017)], they proposed ABNO architecture for multi-layer, multi-domain network orchestration, in which topology updating is performed in a proactive manner.

The protocols for updating the TS can be classified into stateful and stateless protocols [Zhang *et al.* (2021b)]. Stateful protocols maintain all the controllers synchronized at all times with the orchestrator's view of the network. Whenever a change happens in a layer, such as a link failure, the controller of the layer sends a message to the TS to update. In the stateless protocol, the controllers send messages periodically to the TS [Botelho, Ribeiro, Ferreira, Ramos & Bessani (2016)].

Due to the unreliable nature of network communications, implementing a stateful protocol in an MLN is very challenging [Tirehkar, Nguyen & Cheriet (2023)]. Since failures occur very often in this kind of network stateful protocol generates remarkable overheads for updating the TS [Qin, Poularakis, Iosifidis & Tassiulas (2018)] which may be large in some networks such as

wireless networks with limited capacity in-band control channels [Lin, Wang, Akyildiz & Luo (2018)]. Therefore, a stateless protocol is more practical for a complex MLN.

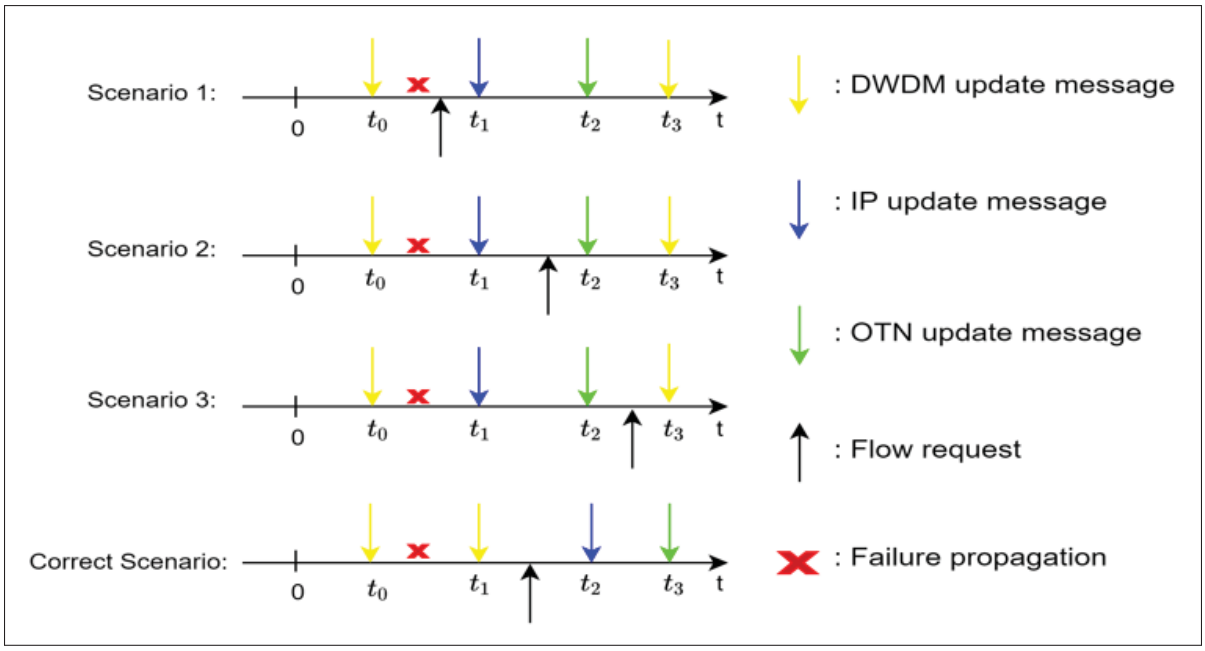


Figure 0.3 Different flow request and updating message times

Problem statement

Through a stateless protocol, controllers update the orchestrator periodically. Such a stateless protocol may result in an inconsistent view of layers, in the interval of time between two consecutive messages from controllers to the orchestrator. Inconsistency can negatively influence the orchestrator's performance. For example, assume the virtual link A-D in the IP layer fails due to an outage in the physical link J-K in the DWDM, as illustrated in Fig. 0.2(b).

As depicted in Fig.0.3 if the stateless orchestrator receives a request for routing a flow from A to D before receiving updates from the controllers (e.g., Scenario 1 in Fig.0.3), it will compute a path based on its current view of the network (A-D and J-K links are all still available), therefore the orchestrator will choose the A-E-I-J-K-G-D path for routing the traffic. This decision will no longer be appropriate because of the outage in the physical link J-K. However, if the orchestrator

had been synchronized with the controllers, it would route traffic over the A-E-F-G-D path. A stateless protocol faces the challenge of deciding at what rate each controller updates the TS.

The order of sending the updates by the controllers is substantial to keep the orchestrator synchronized. To illustrate the effect of the ordered updates on the orchestrator's performance, we assume the orchestrator receives a flow request in different scenarios as in Fig. 0.3 and compare the orchestrator's decisions. In Scenario 2, the orchestrator is updated by the IP layer before receiving the flow request to route a flow from A to D. In this scenario, the orchestrator is aware of the failure in the physical path I-J-K, due to the virtual failure in the IP layer as illustrated in Fig. 0.2(b), however, it is not obvious that failure of I-J or J-K caused failure on the I-J-K path. Therefore, the orchestrator will choose the OTN layer to route the traffic. However, if it was updated by the DWDM layer (e.g. correct scenario of 0.3), it could be aware of the availability of the I-J link and use both OTN and DWDM layers to route the traffic.

Research questions

Using the stateless protocol to update the orchestrator periodically causes inconsistency between the orchestrator and the controllers. In order to reduce this inconsistency we need to address the following question:

- **Research Question (RQ).** What is the synchronization message frequency from controllers to the orchestrator to minimize the inconsistency between the orchestrator and MLN's state?

Objectives of the thesis

Our goal in this thesis is to design an efficient ordered updating mechanism to minimize the inconsistency between the orchestrator and the controllers. We investigate the impact of the synchronization rate from controllers to the orchestrator, on the orchestrator's performance. In addition, our mechanism can avoid confusion in the orchestrator and satisfy the constraints of

the control plane in terms of bandwidth, for the case of single link failure in the MLNs. The main objective can be divided into three sub-objectives (SO):

- **SO1.** Propose an approach of updating the orchestrator's view of the network by controllers. This approach takes into account the order of updates by controllers according to the mapping function and the failure propagation in the MLN.
- **SO2.** Formulate a mathematical optimization model based on the updating approach proposed in SO1 to minimize the inconsistency between the controllers and the orchestrator in the MLN, taking into account bandwidth limitation and failure propagation time.
- **SO3.** Propose an efficient algorithm to solve this optimization problem in near real-time. Consider the high complexity of the optimization model, and increase the scalability of the problem.

Thesis organization

This thesis includes an Introduction, three chapters, and a conclusion.

The Introduction includes a briefing on orchestration in MLN. It also contains motivations for this thesis followed by the thesis objectives.

In Chapter 1 we review the related work on the orchestration in MLN. We also review the protocols that the orchestrator can get notifications from the controllers. Finally, we review the failure propagation in hierarchical MLN.

Chapter 2 presents our methodology to achieve the thesis objectives. It contains our proposed method, a system model based on the proposed method, and a mathematical formulation to optimize the updating message rate from controllers to the orchestrator. In addition, it contains two algorithms for solving the optimization problem in near real-time.

In Chapter 3 we present the experimental setup for validating our proposed method. In addition, we present the numerical results by using a testbed and simulations.

The conclusion summarizes the thesis findings and presents possible future work.

CHAPTER 1

LITERATURE REVIEW

In this chapter, we investigate the available solutions and techniques for updating the orchestrator in an MLN. We also review failure propagation in a hierarchical MLN, and how it affects the order of updating messages from controllers to the orchestrator. Table 1.1 summarizes our literature review on updating the orchestrator in a hierarchical MLN. It contains state-of-the-art on the topics such as SDN orchestration in MLNs, stateful and stateless protocols for updating the orchestrator, failure propagation in an MLN, synchronization rate in a logically-centralized network, and other related approaches.

Regarding our research question, the next section shed a light on the existing updating solutions for MLN. We investigate the pros and cons of the existing works that lead to a pathway for our proposal.

1.1 Failure propagation

When a failure occurs in an underlying layer of MLN, it can propagate to the upper layers based on the network's mapping function. In [Savi & Siracusa (2018)] the authors considered an MLN with an IP/MPLS packet layer, a transparent (DWDM) optical layer, and an SDN orchestrator. Links in the IP layer can be virtual by mapping to the physical links in the DWDM layer. Due to the lack of coordination between the layers, and the lack of ability to provide the application's services to the optical layer, they propose a novel auxiliary-graph-based application-aware service provisioning algorithm. Their proposed algorithm allows to reactively compute new paths in case of failure propagation in the network by occurring a failure in the optical layers, and failure occurrence in the IP/MPLS layer.

In [Williams & Musolesi (2016)] the authors studied spatio-temporal networks by considering how events propagate in the network. They proposed a model of spatio-temporal paths in time-varying spatially embedded networks. They considered a time-varying network as a time-ordered

sequence of graphs. Time intervals have the same size. they considered failure propagation as a function of the physical distance between two nodes and the speed of transmission.

In [Chattopadhyay, Dai *et al.* (2019)] the authors studied the methods to optimize the interlinking structure of multilayer interdependent networks with cost constraints in order to increase network robustness against node failures. Instead of using popular methods such as branching process analysis or supra-adjacency matrix representation, this study proposes a framework that utilizes surrogate metrics to construct interlinks that improve network robustness. The authors focused on three failure propagation mechanisms and proposes metrics to track network robustness for each mechanism.

In our problem, we studied failure propagation in multilayer networks in case of the occurrence of a link failure in the underlying layers. These failures in the physical part of the network propagate to the upper layers if there are any entities in the upper layers that are mapped to the failed link. We take into account the time that a failure takes to propagate from an underlying link to the links in the upper layers by considering the path between them.

The critical problem with failure occurrence and propagation in MLN is how the controllers with failures in their domain update the orchestrator about their current state. computer networks are a crucial component of infrastructure, and they are expected to meet high standards in terms of accuracy, accessibility, and performance. they should also be flexible and adaptable to allow for fast updates, such as changes in policies, traffic surges, or system failures. [Foerster, Schmid & Vissicchio (2018)] presents a comprehensive review of mechanisms and protocols that facilitate fast and consistent updates to computer networks. The authors discuss the desirable consistency properties that should be ensured throughout a network update, the algorithmic techniques required to achieve these properties, and the implications for the speed and cost of updates.

The next sections investigate the existing protocols that controllers of MLN can use to update the orchestrator about their domain's current state.

1.1.1 Stateful protocol

The stateful protocol ensures that the orchestrator has an updated view of the current state in each domain. When a change happens in a domain, that domain's controller updates the orchestrator about the change immediately. In an SDN orchestration structure, the orchestrator coordinates the resource provisioning across multiple domains and technologies based on its global view of the network. This implies that the orchestrator should maintain the new view of the network when a change happens in a domain [Botelho *et al.* (2016)]. In this regard, Botelho et. al was motivated by the need for a stateful protocol to assure correct network policy enforcement. They proposed an architecture, contrary to alternative designs, their central controller is consistent, fault-tolerant, and guarantees that applications work on a consistent view. The main concern of their approach is the overhead required to guarantee consistency on fault-tolerant, which may limit the scalability. They proposed several optimization techniques to reduce the effect of overheads on the application's performance.

[Katta, Zhang, Freedman & Rexford (2015)] proposed Ravana, which is a platform for SDN controllers that provides a fault-free centralized controller abstraction to manage applications. Rather than simply maintaining consistency in the controller's state, Ravana treats the entire event-processing cycle (which includes event delivery from switches, event processing on controllers, and command execution on switches) as a transaction. Ravana ensures that transactions are executed as a whole, either all or none of the components and that they are ordered across replicas and executed only once throughout the system. As a result, Ravana can handle switch states accurately without relying on rollbacks or repeated execution of commands. Switches update the controllers consistently. Their method makes less overhead than strongly consistent protocol. However, in our structure, there are multiple controllers, in which each controller controls a set of switches in its domain and updates the orchestrator of its domain's current state.

1.1.2 Stateless protocol

The stateless protocol orchestrator receives updating messages from the controllers periodically. When a change occurs in a domain, that domain's controller updates the orchestrator about the change by the next message that will be sent to the orchestrator at the end of the time interval. In [Mayoral, Vilalta, Munoz, Casellas & Martánez (2015)], they proposed a multi-domain SDN orchestration and analyzed its performance on multi-layer end-to-end services in a multi-domain network scenario. They investigated topology recovery by both stateful and stateless protocols. Although the stateless protocol can increase the inconsistency and blocking probability when a change occurs between two updating messages, they chose this method to reduce the delay to answer a service request.

In [Levin, Wundsam, Heller, Handigol & Feldmann (2012)], the authors considered a logically centralized network and investigated the reliability and scalability of the network according to different synchronization dissemination methods. They evaluate the performance of a load balancer application according to an inconsistent global view of the network. In addition, they take into account the trade-off between the application performance and network state overhead and the complexity of application logic vs robustness to inconsistency in the underlying distributed SDN state.

[Zhang *et al.* (2019)] proposed a synchronization policy design between the controllers based on deep reinforcement learning. Many current approaches focus on resolving inconsistencies between controllers' network views to eliminate anomalies. They formulated the controller synchronization problem as a Markov decision process (MDP) and leveraged reinforcement learning techniques along with deep neural networks (DNNs) to train a sophisticated and scalable controller synchronization policy called Multi-Armed Cooperative Synchronization (MACS). The primary objective of MACS is to maximize the performance benefits of controller synchronization. However, in our work, we consider the hierarchically structured network, in which failures propagate within layers. In addition, in our proposed method we find the exact number of updates by each controller to reduce the inconsistency.

In [Guo *et al.* (2014)], the authors investigated synchronization in a multi-controller multi-domain SDN network. They mentioned if controllers update the logical view of the network frequently when a failure occurs in the network, the network may experience high synchronization overhead. In addition, the state of the controllers may become desynchronized between two consecutive synchronizations, which could lead to forwarding loops and black holes. To address this issue, they proposed a new scheme called Load Variance-based Synchronization (LVS) that improves load-balancing performance. Unlike existing schemes, LVS conducts effective state synchronizations among controllers only when a load of a specific server or domain exceeds a certain threshold. This significantly reduces the synchronization overhead of controllers. Their simulation results demonstrate that LVS achieves loop-free forwarding and good load-balancing performance with much less synchronization overhead than existing schemes that use PS-based synchronization. However, their method only focused on the performance of load-balancing applications. However, in our thesis, we consider the overhead based on our resources, and we reduced the inconsistency between the orchestrator and controllers, which reduced desynchronization between the orchestrator and the controllers. Moreover, our method is not limited to one application.

In [Mirkhanzadeh *et al.* (2018)] the authors studied MLNs which are managed by a hierarchical SDN orchestrator. There is a Resource Management (RM) module in their study that serves as an interface/adapter for the orchestrator and is responsible for sending and receiving messages to and from the controllers. It uses REST APIs provided by three SDN controllers and the RESTCONF Optical Plug-in to communicate with both electrical packet and optical circuit domains. The RM module periodically discovers each layer's network topology, creates a global view of the network for the orchestrator, and stores it in the Database. Apart from these functionalities, the RM module also handles path setup requests originating from the service provisioning module and uses REST APIs and RESTCONF interfaces to apply the necessary network equipment configuration commands. Additionally, it fetches network alarms caused by fiber outages and sends messages to the fault handler module to notify it for appropriate counteractions. However, they did not mention if a failure occurs between two consecutive topology discoveries, how the

orchestrator will route the traffic. In our thesis, we studied this problem and proposed a method to reduce this inconsistency between the orchestrator and network state.

As the above studies mentioned the inconsistency between the orchestrator and the controllers in the stateless protocol can have negative effects on the performance of the orchestrator. To overcome this in the next step we discussed other methods which contain the dynamic protocol, and investigated the frequency of synchronization messages.

1.1.3 Dynamic protocol

The authors of [Sakic, Sardis, Guck & Kellerer (2017)] proposed an adaptive dynamic synchronization method, which is a combination of stateful and stateless protocols. This approach is based on the severity of each failure, and the cost of its consequences. They divided the failures into two groups, strict and not strict based on the failure's impact on the network. When a strict failure occurs which affects a large portion of network entities and infrastructure in a controller's domain, the controller has to send update messages to other controllers to inform them about the failures. However, if the occurred failure is not strict the controller can keep updating other controllers periodically. They showed this method improved the performance of network applications and reduced application inefficiency, and it is scalable. However, they did not consider the effect of the synchronization rate on the performance of the applications.

The stateless protocol can have negative impacts on the network. When a failure occurs in a domain in an interval of time between two update messages, the orchestrator is not aware of that until the next interval. For example, Fig.1.1 shows the time of update messages from layer i to the orchestrator, and a failure occurred after the first update message however, the orchestrator will be updated about that at t_1. To overcome this inconsistency between the controllers and the orchestrator some papers work on sending additional messages within each interval by considering the resource limitations. In [Poularakis *et al.* (2019)], the authors investigated synchronization in a logically-centralized network. They considered a distributed SDN controllers architecture, in which each controller is responsible for its domain. Each

controller disseminates synchronization messages to the other controllers when a change occurs in its domain (i.e. link failure). Disseminated messages contain the controller's view of the current state of its domain (i.e. available resources). They considered that controllers disseminate synchronization messages periodically. To reduce the inconsistency among the controllers they investigated policies among them and focused on the synchronization frequency rate on the performance of network applications.

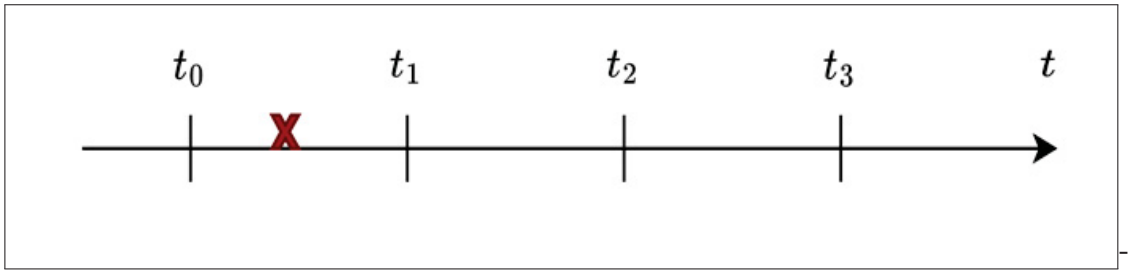


Figure 1.1 failure occurs in layer I between two updated messages

The following section investigates protocols to apply the aforementioned methods to update the orchestrator in MLN.

1.2 Topology discovery in MLN

In an MLN each layer is a technology with unique functionality, in which an orchestrator is required to use all of these technologies [Mirkhanzadeh *et al.* (2018)]. The orchestrator makes its decision based on the last topology gathered by TS. Orchestrator creates its network topology by cooperation with the domain's controllers. There are different protocols that the orchestrator can use for this cooperation. Domain controllers can have their own RESTFUL APIs to send their current state information to the other applications however Network Configuration Protocol (NETCONF) is one of the protocols that controllers can use. NETCONF defines a mechanism that controllers can be managed by the orchestrator. For instance, the controller's configuration data can be retrieved, uploaded, and manipulated by the orchestrator [Enns, Bjorklund, Schoenwaelder & Bierman (2011)]. RESTCONF is NETCONF's RESTful-based

version, which can be applied to exchange messages. YANG (Yet Another New Language) makes NETCONF and RESTCONF more configurable.

1.2.1 YANG modeling language

YANG is a data modeling language, designed to model configuration, operational data declarations, and notifications. Due to its flexibility, and availability of tools, it has become the data modeling language for multiple network control and management aspects [Bjorklund (2016)]. YANG language is using encapsulation of containers and lists to structure the data which is expressive.

In [Casellas *et al.* (2018)], the authors proposed control, management, and orchestration systems in multi-layer multi-domain networks. They investigated efficient methods for data collection from different sources in a network to monitor the infrastructure. This method requires the adoption of interfaces and development. Monitoring the status of entities in a large-scale network, configuring programmable pipelines, and enabling performance monitoring are precise functional requirements for the interfaces. They proposed YANG notification mechanism that allows the network operator to receive notifications from clients, configure parameters for filtering the notifications, and to choose the method of receiving the notifications such as strongly consistent or eventually consistent.

There are two subscription methods for using the YANG modeling language by the orchestrator to receive notification updates from the controllers. These methods are named periodic subscriptions (stateless), and on-change subscriptions (stateful). By periodic subscription, the orchestrator receives updates from each controller periodically according to some time interval. However, by using an on-change subscription, whenever a change occurs in the domain of a controller, it updates the orchestrator of this change and available resources of its domain (i.e. available links, available nodes) [Mayoral *et al.* (2020)].

However, no prior work has investigated the rate of synchronization messages from each controller to the orchestrator in a hierarchical multi-layer multi-domain (MLMD) network. In

addition, We take into account the order of updating by controllers due to failure occurrence, failure propagation, and mapping function.

Table 1.1 Literature review

Reference	Feature	Pros	Cons
Yang data modelling Enns *et al.* (2011) Casellas *et al.* (2018) Mayoral *et al.* (2020)	Standard modelling language	Configure parameters, filter messages , and send notifications	Not suitable for every type of applications
Sateful protocol Botelho *et al.* (2016) Katta *et al.* (2015)	Strongly consistent view of the network	The orchestrator and controllers are synchronized all the time	Limited scalability and generate significant overheads
Stateless protocol Mayoral *et al.* (2015) Levin *et al.* (2012) Sakic *et al.* (2017) Zhang *et al.* (2019) Mirkhanzadeh *et al.* (2018)	Update the orchestrator periodically	Less overheads in complex network	Inconsistency between the orchestrator and controllers
Synchronization rate Poularakis *et al.* (2019) Sakic *et al.* (2017)	More update messages in an interval time	Reudce the inconsistency between the orchestrator and controllers	Not considering correlation between domains

CHAPTER 2

METHODOLOGY

This chapter presents the methodology of our work. We first introduce the architecture, which includes illustrations of MLN. Then, our proposed framework for the synchronization of the orchestrator and controllers is presented.

2.1 Architecture

2.1.1 Multi-layer multi-domain orchestration

In this thesis, we consider a model representing a stateless MLN consisting of L layers, in which each layer is a domain and an orchestrator. Each layer has a controller that manages that layer and has the layer's information (e.g., available resources, failed entities).
The orchestrator is aware of the whole network's state until the last update that TS received from all layers. The orchestrator can decide how to coordinate multiple layers and how to send the traffic through different layers from a source to a destination based on the network's available resources (e.g., available links) provided by TS. Table 2.1 summarizes the notations and variables used in this thesis.

2.1.2 Failure propagation

Due to the hierarchical architecture of MLN, a failure in an underlying layer can cause failures in upper layers, which is called failure propagation. Therefore, we classify the failures into two categories. The first category includes the failures that happen independently of the other failures in the lower layers. The second category contains the failures that happen as a result of other failures in the lower layers, which means these failures happen in the dependent part of the layer. Let the mapping function $m_{k,i}$, denotes the mapping relation between upper layer k and lower layer i. If layer k is mapped to layer i, $m_{k,i} = 1$, otherwise $m_{k,i} = 0$. And, $U(i) = \{k|m_{k,i} = 1\}$ shows a set of layers that are mapped to layer i. So, when a failure happens in a link in an

Table 2.1 Notations

Notation	***Description***
λ_i	State change rate of the independent part of controller i.
t	Time window length.
R_i	Represents the maximum number of additional messages from controller i, to the TS.
c_i	Resource cost of sending a message from controller i to the TS.
B	The available network resource (Bandwidth).
w_{ij}	Percentage of links in the independent part of layer j (lower layer), that links from layer i (upper layer) are mapped to them.
$p_i(a, n)$	Failure probability in layer i, in the interval with length n after the a_{th} message and before $(a + 1)_{th}$ message to the TS.
$m_{k,i}$	Shows the mapping relation between layer k and layer i.
$U(i)$	Set of layers that are mapped to layer i.
$l(i)$	Set of links of layer i.
s_{ik}^{bound}	Failure propagation speed in the boundary link between layer i and layer k.
$t_i(a)$	Time of the a_{th} update from layer i to the TS.
$T_i^k(a, b)$	Failure propagation time from link $a \in l(i)$ to the link $b \in l(k)$.
$T_{prop}(i, k)$	Failure propagation time from layer i to layer k.
Variables	***Description***
x_i	The number of additional messages within each time window, between controller i and the TS.

underlying layer, by finding the distance between the failed link and the virtual links that are mapped to it, we can calculate the failure time at each of those links.

The failure propagation time is the time that takes a link failure in the underlying layer propagates to the links in the upper layers [Salama, Ezzeldin, El-Dakhakhni & Tait (2022)], which are mapped to the failed link. We calculate the propagation time from an underlying layer to the upper layers by considering the Spatio-temporal effect [Williams & Musolesi (2016)]. Let $l(i)$ represent a set of links of layer i, we assume the failure propagation speed in all links of each layer is the same and denote this link propagation speed of layer i by s_i. In addition, we denote the propagation speed of the boundary link, between layer i and layer k by s_{ik}^{bound}. We calculate the shortest path from the physical link $a \in l(i)$, to the virtual link $b \in l(k)$ by using Dijkstra's algorithm, and show the time of this propagation by $T_i^k(a, b)$ Salama *et al.* (2022).

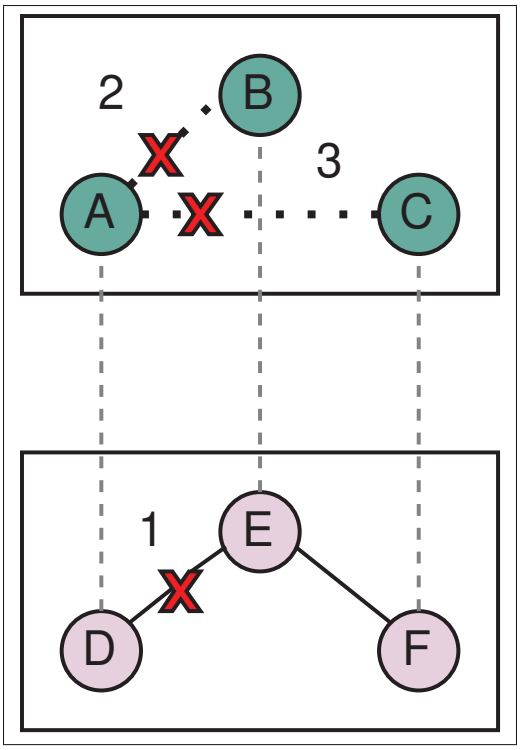


Figure 2.1 Propagation time between layers

We define the failure propagation time between layer i and layer k, as the maximum link propagation time between layer i and layer k, which is denoted by: $T_{prop}(i,k)$=$max\{T_i^k(a,b)|\forall a \in l(i), b \in l(k)\}$. For instance, in Fig. 2.1, virtual link A-B(2) is mapped to the physical link D-E(1), and virtual link A-C(3) is mapped to the set of physical links D-E-F, the failure propagation time between these two layer is: $T_{prop}(0,1)$=$max\{T_0^1(1,2), T_0^1(1,3)\}$.

2.2 System modeling

We denote the state change rate of the independent part of the controller i by λ_i. In addition, changing network conditions in the independent part of each layer can be modeled by an independent Poisson process with a state change rate λ_i. We consider a period T, which is then divided into time windows. Each time window is denoted by t. Moreover, assume that all controllers are sending messages to the orchestrator at the beginning of each time window. According to the Poisson process model, the probability that the state of the independent part of

the controller i does not change is $e^{-\lambda_i t}$, where t is the time window length. The probability of failure occurrence independently in layer i is $1-(e^{-\lambda_i t})$. The probability of independent failure occurrence in all the layers is given by:

$$\sum_{i=0}^{L-1}(1 - e^{-\lambda_i t})$$

We define the weight of mapped entities from layer i (upper layer) to layer j (lower layer), by the percentage of independent entities in layer j that corresponds to a mapping from layer i. Such weight is denoted by:

$$w_{ij} = \begin{cases} 0 < w_{ij} \leq 1 & \text{if layer } i \text{ is mapped to layer j} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

In addition, the probability of failure occurrence in the dependent part of layer i due to the occurrence of a failure in layer j, which is one of the i's lower layers, is: $(1 - e^{-\lambda_j})w_{ij}$. Therefore, the probability of failure occurrence in the dependent part of layer i due to the propagation of failures in the underlying layers is:

$$\sum_{j=0}^{i-1}(1 - e^{-\lambda_j t})w_{ij}$$

Because only a higher layer can be mapped to a lower layer, this failure probability can be written as follows:

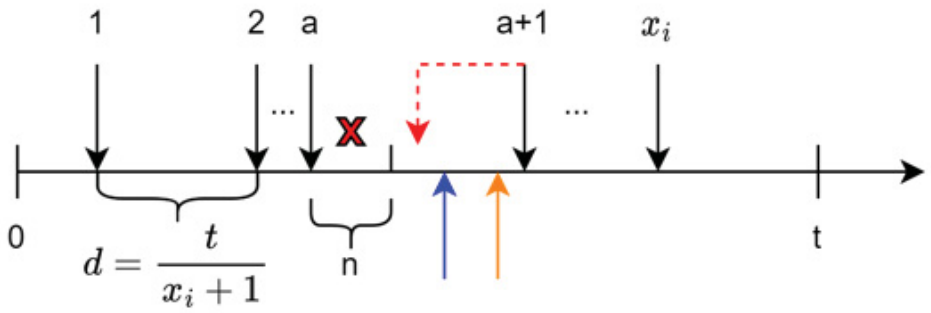
$$\sum_{j=0}^{i-1}(1 - e^{-\lambda_j t})w_{ij} = \sum_{j=0}^{L-1}(1 - e^{-\lambda_j t})w_{ij} \tag{2.2}$$

To update the orchestrator's view of the network, the controllers have the option to send additional messages to the orchestrator within each time window. We denote these extra messages that are sent from controller i to the orchestrator by x_i (e.g., Fig. 2.2). And we denote the maximum

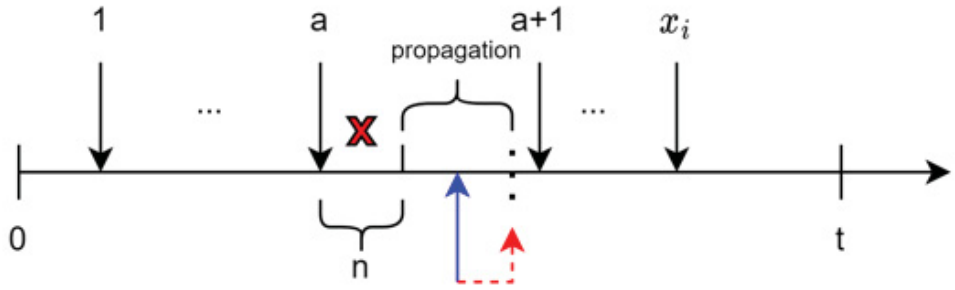
number of messages that controller i ($\forall i \in \{0, 1, 2, ..., L-1\}$) can send to the orchestrator by R_i.

$$x = (x_0, x_1, ..., x_{L-1}) \tag{2.3}$$

$$0 \leq x_i \leq R_i : \forall i \in \{0, 1, ..., L-1\} \tag{2.4}$$



a) Update the orchestrator sooner than upper layers, to avoid confusion in case of occurrence of a failure in an underlying layer.



b) Update the orchestrator after the required time for failure propagation to convey correct information about the layer's state.

Figure 2.2 Failure propagation and mapping between the layers

Sending a message from each controller to the orchestrator consumes the network's bandwidth. Bandwidth consumption can be significant in resource-constrained environments [Huang, Cui, Zhang, Chu & Lin (2020)]. We define c_i as the resource cost (e.g., bandwidth consumption) of

sending each message from controller i to the orchestrator. Let B denote the network's available bandwidth, the total bandwidth consumed to send the update message to the orchestrator cannot exceed the total available bandwidth B:

$$\sum_{i=0}^{L-1} x_i c_i \leq B \tag{2.5}$$

Therefore, the time interval between two consecutive additional messages that are sent from layer i to the orchestrator is $\frac{t}{x_i+1}$ as depicted in Fig. 2.2(a). We call the first physical link failure the main failure and the consequent failures as virtual failures. If one of the layers with virtual failure sends a message to the orchestrator sooner than the layer with the main failure, the orchestrator will get confused, and cannot build a correct view of the available resources.

For instance, in Fig. 2.2(a), a failure happens within an interval with length n, after the a_{th} update from layer i, if upper layers (blue and orange arrows) update the orchestrator after the failure and sooner than $(a+1)_{th}$ update from layer i, the orchestrator will not find the correct state of layer i. So, as the red dashed arrow shows, layer i should send $(a+1)_{th}$ message to the orchestrator before its upper layers which are mapped to it. We denote the probability of the occurrence of an independent failure in layer i, in an interval with length n after a_{th} message and before $(a+1)_{th}$ message to the orchestrator by $p_i(a,n)$, which can be calculated as:

$$p_i(a,n) = (1 - e^{\frac{-\lambda_i t}{x_i+1}}) \times \frac{n}{\frac{t}{x_i+1}} \tag{2.6}$$

Let $t_i(a)$ denotes the time of the a_{th} message from layer i to the orchestrator, it can be calculated as:

$$t_i(a) = a(\frac{t}{x_i+1}) \tag{2.7}$$

Let H(k), $\forall k \in U(i)$, shows all the messages from layer k that have been sent to the orchestrator after the failure and before the layer i's $(a+1)_{th}$ message. The inequality (2.8) states that after sending the a_{th} message to the orchestrator by layer i, if a failure happens in that layer, its

controller must send the next message ($(a+1)_{th}$ message) to the orchestrator before its upper layers that are mapped to it.

$$\begin{gathered} p_i(a,n) \times t_i(a+1) \leq t_k(b) \\ \forall a \in \{0,1,...,x_i-1\}, \forall k \in U(i), \forall b \in H(k) \end{gathered} \tag{2.8}$$

When a failure occurs in layer i, it takes $T_{prop}(i,k)$ to propagate from layer i to layer k. So, if layer k updates the orchestrator before the failure propagation time, its controller is not aware of the failure that will occur in the virtual link and will not send the correct information to the orchestrator. For example, in Fig. 2.2(b), the time of updating the orchestrator by the upper layer (blue arrow) is not correct due to the propagation time. However, as the dashed red arrow shows, if layer k sends the updating message to the orchestrator after the propagation time, it can convey correct information to the orchestrator. Therefore, constraint 2.9 indicates that if a failure occurs in an interval with length n after the a_{th} update from layer i, the upper layers that are mapped to layer i, must update the orchestrator after the propagation time.

$$\begin{gathered} p_i(a,n) \times t_i(a) + n + T_{prop}(i,k) \leq t_k(b) \\ \forall a \in \{0,1,...,x_i-1\}, \forall k \in U(i), \forall b \in H(k) \end{gathered} \tag{2.9}$$

By sending more messages in each time window, in case of occurrence a failure we can inform the orchestrator sooner, and the orchestrator can make better decisions. We can effectively reduce the time window by a factor of the number of additional messages. As we showed in Fig. 2.2(a), we can reduce the inconsistency time between the controllers and the orchestrator from t to $\frac{t}{x_i+1}$. Therefore, the probability of occurrence of a failure decreases from $1-e^{-\lambda_i}$ to:

$$1 - e^{\frac{-\lambda_i t}{x_i+1}}$$

So, the probability of occurrence of failures in layer i is the sum of the failure probability in the dependent and independent part of layer i, namely:

$$\Omega\left(x_i\right) = (1 - e^{\frac{-\lambda_i t}{x_i+1}}) + \sum_{j=0}^{i-1}(1 - e^{\frac{-\lambda_j t}{x_j+1}})w_{ij} \tag{2.10}$$

By considering equation (2.2) we can rewrite equation (2.10) as:

$$\Omega\left(x_i\right) = (1 - e^{\frac{-\lambda_i t}{x_i+1}}) + \sum_{j=0}^{L-1}(1 - e^{\frac{-\lambda_j t}{x_j+1}})w_{ij} \tag{2.11}$$

Finally, the probability of failure occurrence in the network between two consecutive messages from each layer to the orchestrator is:

$$\Omega\left(x\right) = \sum_{i=0}^{L-1}(1 - e^{\frac{-\lambda_i t}{x_i+1}}) + \sum_{i=0}^{L-1}\sum_{j=0}^{L-1}(1 - e^{\frac{-\lambda_j t}{x_j+1}})w_{ij} \tag{2.12}$$

Our goal is to minimize this objective function which is showing the failure probability in inconsistency time between the orchestrator and controllers. Therefore, the optimization model for updating the orchestrator's view of the network can be formulated as follows:

$$\min_{x} \quad \Omega\left(x\right) \tag{2.13}$$

$$\text{s. t.} \quad (2.4), (2.5), (2.8), (2.9)$$

$$0 \leq w_{ij} \leq 1, \forall i, j \in \{0, 1, ..., L-1\} \tag{2.14}$$

2.3 Algorithmic Solution

The objective of our optimization problem is $\Omega(x)$, which is the summation of exponential functions. According to the convexity of exponential functions, and the summation of convex functions is convex, therefore $\Omega(x)$ is a convex function. We can show that this problem is

NP-Hard by reducing it to the Knapsack problem (which is NP-Hard), specifically our object is reducible to Multiple-Choice Knapsack (MCK) problem [Poularakis *et al.* (2019)].

To solve this problem in a timely fashion, we propose two algorithms in this section. Algorithm 1, named Multi-Layer Network Orchestrator Updating (MLNOU), to approximates the optimal solution of the aforementioned optimization problem in (2.13). The MLNOU algorithm is based on the Simulated Annealing algorithm which uses stochastic global search.
Using randomness for the search process makes this algorithm a good candidate for our problem because the nonlinear objective function does not allow other local search algorithms to work appropriately.

We set the number of Iterations and T as the input for Algorithm 1, and it generates vector x_{opt} and Ω_{opt} as the output. In the third line of Algorithm 1, the initial function randomly generates a feasible vector x, and then we calculate the objective function $\Omega(x)$ for this vector. In addition, we set the optimal value Ω in line 6, then we update the optimal vector and optimal value in the next steps. For each iteration in line 7, use the $calculate()$ and $neighbor()$ functions to compute respectively a new $T_{current}$ and a new x_{test}, then calculate the objective function Ω_{test}. If the objective function is improved by a new vector, we accept the new vector and then update x and Ω in lines 12 and 13. If the objective is smaller than the optimized one, we replace the optimized objective with the new objective and vector x with the new vector in line 15 and line 16.

On the other hand, if the objective function gets worse, in line 18 we calculate the acceptance probability which is the difference of ω and ω_{test} divided by $T_{current}$ and then call the $rand()$ function to generate a random number between 0 and 1. If the acceptance probability is greater than this random number, the new vector and its objective function are accepted, and we update x and Ω with new values in line 19 and line 20. We keep doing this for a sufficiently large number of iterations.

Algorithm 2.1 MLNOU

Input:Iterations, T
Output: x_{opt} and Ω_{opt}
$x \leftarrow Initial()$
$\Omega \leftarrow \Omega(x)$
$x_{opt} \leftarrow x$
$\Omega_{opt} \leftarrow \Omega(x)$
for $i = 1, 2, \ldots, Iterations$ **do**
  $x_{test} \leftarrow neighbor(x)$
  $T_{current} = calculate(i, T)$
  $\Omega_{test} \leftarrow \Omega(x_i)$
  if $\Omega_{test} \leq \Omega$ **then**
    $x \leftarrow x_{test}$
    $\Omega \leftarrow \Omega_{test}$
    if $\Omega_{test} \leq \Omega_{opt}$ **then**
      $x_{opt} \leftarrow x_{test}$
      $\Omega_{opt} \leftarrow \Omega_{test}$
    end if
  else if $exp(\frac{\Omega - \Omega_{test}}{T_{current}}) \geq rand()$ **then**
    $x \leftarrow x_{test}$
    $\Omega \leftarrow \Omega_{test}$
  end if
end for

We refer to the second algorithm as Multi-Layer Network Stochastic Greedy (MLN-SG) and summarize it in Algorithm 2. MLN-SG is based on the stochastic greedy algorithm. In each iteration, the algorithm randomly chooses a subset of solutions and finds the best solution within the subset. In this process, we generate solutions and the number of solutions is a function of the size of samples (t) from X, and then by comparing these solutions with the optimal one,

update the Ω_{opt} and X_{opt}. In the end, this algorithm generates the best Ω_{opt} and X_{opt} as outputs. In the third line of Algorithm 2, we set the Ωopt equal to infinity, and in lines 4 and line 5 set X_{opt} and E equal to the empty set. In addition, for each iteration, we generate a set of feasible solutions named X (line 7). When this set X is not empty, we randomly select a subset E of these feasible solutions by sampling t random items from X (line 10). Moreover, in line 15, we choose $argmin$ of the selected subset and add it to the set S.

Algorithm 2.2 MLN-SG

1: Input: $t, iterations$
2: Output: X_{opt}, Ω_{opt}
3: $\Omega_{opt} \leftarrow \infty$
4: $X_{opt} \leftarrow \emptyset$
5: $E \leftarrow \emptyset$
6: **for** $i = 1, \ldots, iterations$ **do**
7: $X \leftarrow$ set of feasible solutions
8: $S \leftarrow \emptyset$
9: **while** X is not empty **do**
10: $E \leftarrow$ a random subset of feasible solutions by sampling t random items from X
11: $X \leftarrow X \backslash E$
12: $k \leftarrow argmin_{s \in E} \Omega(s)$
13: $S \leftarrow S \cup k$
14: **end while**
15: $s_i \leftarrow argmin_{s \in S} \Omega(s)$
16: **if** $\Omega(s_i) \leq \Omega_{opt}$ **then**
17: $\Omega_{opt} \leftarrow \Omega(s_i)$
18: $X_{opt} \leftarrow s_i$
19: **end if**
20: **end for**

2.4 Discussion

In the Introduction, we mentioned the main principles of orchestration in an MLN: synchronization protocols, and failure propagation. Here we explain how inconsistency between the orchestrator and controllers can make problems and affect the orchestrator's performance. In addition, we explained how failure propagation can affect multiple layers and it has negative effects on the orchestrator's performance.

In this chapter, we have presented the research methodology. We explained the architecture of MLN, and how a failure propagates in the hierarchical MLN. In addition, we showed sending more messages in an interval of time between two update messages reduces the inconsistency between the orchestrator and the controllers.
In our method, we investigated different methods of updating and the order of updating and showed how a wrong update can affect the orchestrator's performance.

Then, we proposed an approach to update the orchestrator by each domain's controller, according to the bandwidth limitation and failure propagation in the MLN. We formulated a mathematical optimization model based on our approach. Moreover, we proposed algorithms to solve this problem.

CHAPTER 3

NUMERICAL RESULTS

In this section, we present the experimental results carried out in a national-wide MLN testbed and in a large-scale simulation network, the Coronet network. We compare the proposed MLNOU algorithm with three baselines that have been tested on different network topologies, but by considering their characteristics, comparing MLNOU with them shows how well MLNOU works: 1) MLN-SG which is based on the stochastic greedy algorithm to generate an approximate of the optimal solution, this method is reaching a close to optimal solution in a timely manner, 2) the optimal solution of our problem (2.13) which is obtained by a mathematical solver (CVXPY [Diamond & Boyd (2016)]), and 3) a Homogeneous algorithm in which all the layers update the TS at equal rate [Poularakis *et al.* (2019)].

The metrics used for comparison include: i) the failure probability obtained by the baselines, ii) the bandwidth required for the updates, iii) the state change rate (λ), and iv) the time required (Run Time) to obtain the solution. To avoid randomness in the MLNOU algorithm we calculate the result 100 times and then get the average value taking into account different numbers of iterations and different T. We tried numbers less than and more than 100, but for reducing the effect of outliers and the run-time of the algorithm, 100 is a good trade-off. In addition, we provide an emulation study that will illustrate the impact of updating rate on the performance of the shortest path routing application.

3.0.1 Testbed protocol

We evaluate the performance of the proposed MLNOU and MLN-SG on a nationwide testbed. Our MLN at Telus-Ciena Lab in Edmonton, AB, is a packet-optical network with SDN capabilities as depicted in Fig. 3.1. Our testbed consists of three (IP/OTN/DWDM) layers, and each layer is managed by a controller: MCP for L0 (DWDM), Virtuora for L1 (OTN), and Northstar for L2 (IP). All of these three controllers are connected to an orchestrator (named Synchromedia).

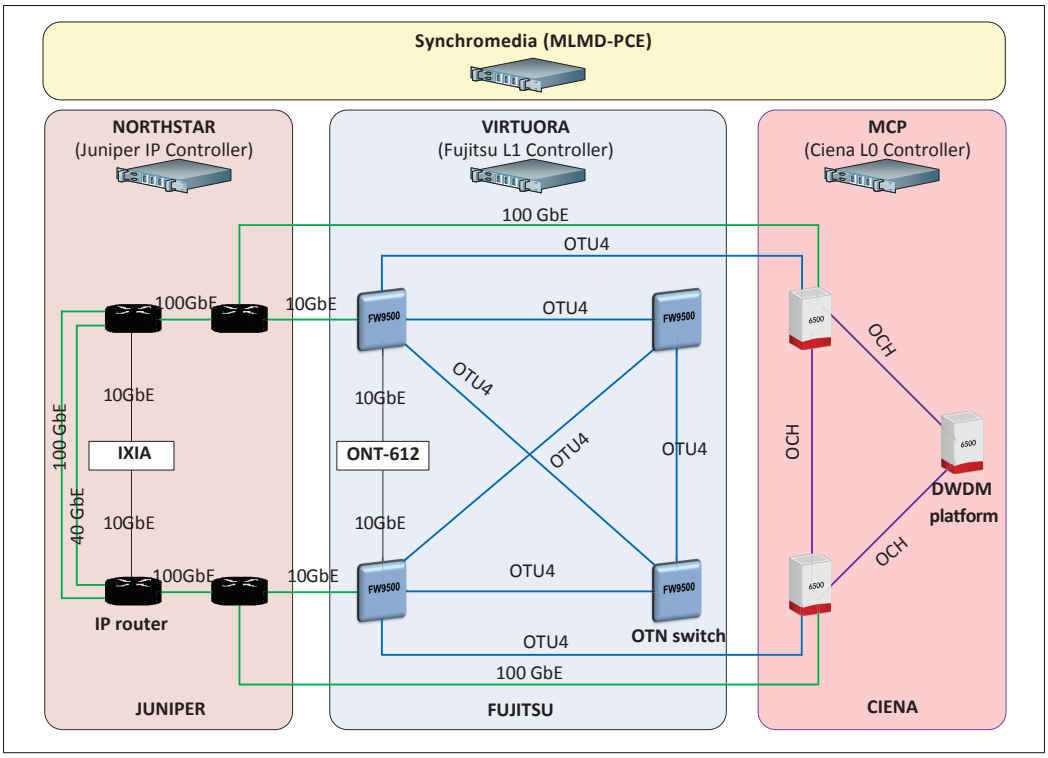
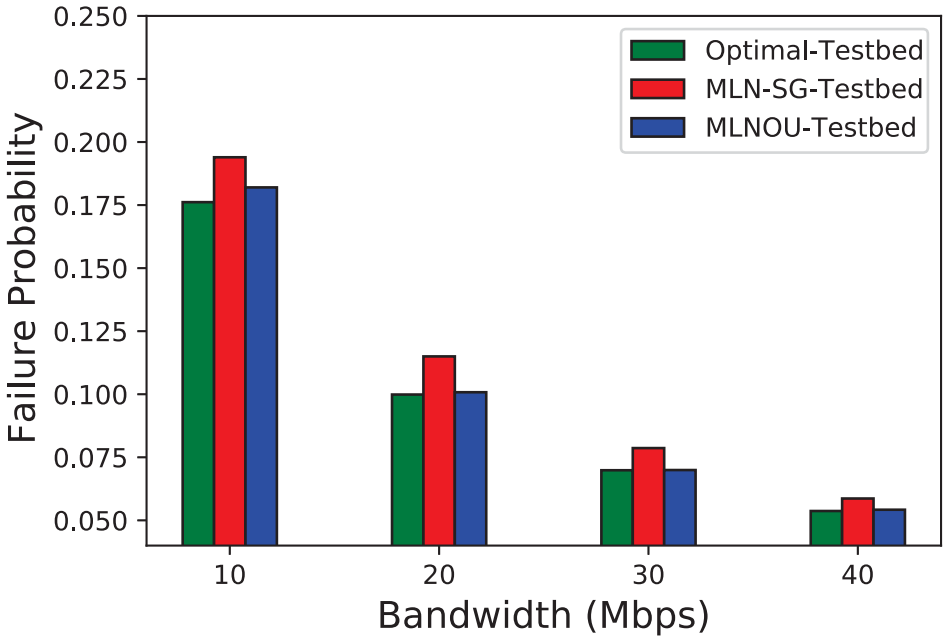


Figure 3.1 The MLN Testbed at Telus-Ciena Laboratory

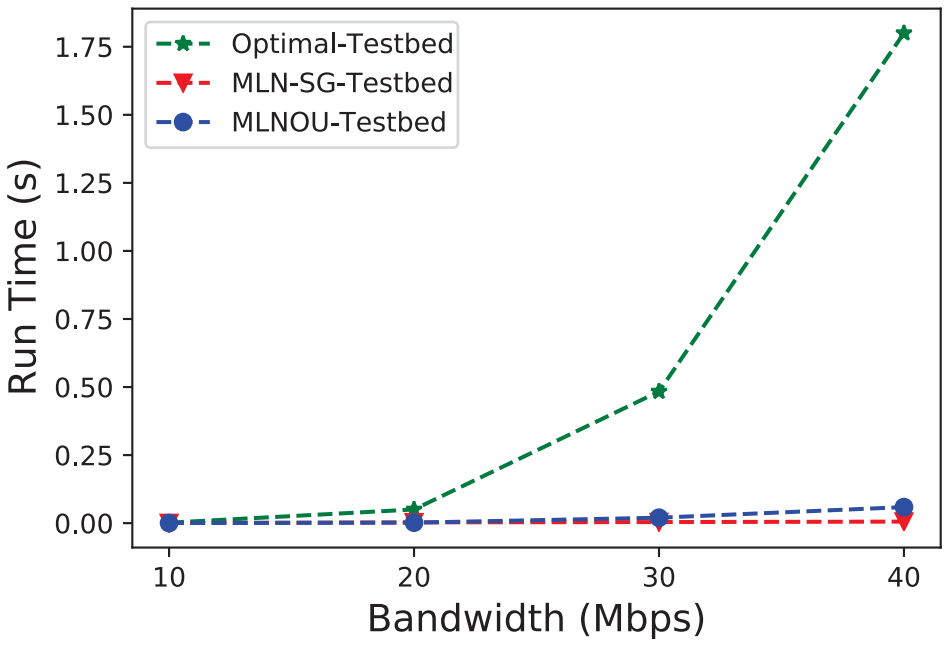
Layer 0 (DWDM) has three Ciena 6500 DWDM platforms, layer 1 (OTN) has four Fujitsu optical switches, and layer 2 (IP) has four Juniper routers [Koulougli *et al.* (2020)].

3.0.2 Results obtained in the testbed

Based on the topology of our testbed if there is an underlying path between two nodes in the IP layer, we assume there is a virtual link between them and is mapped to the underlying path, and failures propagate from underlying links to the upper layer's links. We evaluate the probability of occurrence of a failure in the inconsistency time between the controllers and the orchestrator, achieved by MLNOU, MLN-SG, and the optimal solution by considering different bandwidth constraints. Fig. 3.2 shows the effect of updating rate on the orchestrator's routing performance. We evaluate the minimum failure probability that each algorithm can achieve by using the equation (2.12), according to different bandwidths (B), and different state change rates (λ) of the independent parts of the network. As depicted in Fig. 3.2(a), by increasing the bandwidth, the



a) Performance comparison of the proposed algorithms on the testbed



b) Run Time comparison

Figure 3.2 Performance comparison of the proposed algorithms on the testbed

failure probability decreases for all the baselines, which is caused by sending more messages by the controllers to the orchestrator. In all scenarios, MLNOU is closer to the optimal solution and performs better than the MLN-SG algorithm and in the case of using MLNOU, failure probability is about 10% better than the MLN-SG algorithm. We compare the results up to 40 Mbps bandwidth because it shows how the optimal solution's run time is getting further from other algorithms, and in failure probability comparison after 40 Mbps the differences get less.

As depicted in Fig. 3.2(b), the runtime of all of the baselines increases along with the bandwidth. However, the runtime of the optimal algorithm increases much faster than the other algorithms. The runtime of both MLNOU and MLN-SG algorithms is quite similar and very steady, which shows these algorithms are scalable.

3.0.3 Results obtained in simulations

For the simulated topology, we use the Coronet network, its topology is depicted in Fig. 3.3 [Von Lehmen *et al.* (2015)] with three layers and 60 nodes. We generate a three-layers hierarchical network from this topology as shown in Fig. 3.3, by randomly choosing nodes for each layer (depicted in Fig. 3.4. There are 15 nodes in layer 2, 10 nodes in layer 1, and 35 nodes in layer 0.

In Fig. 3.5, we evaluate the performance of Optimal, MLNOU, MLN-SG, and Homogeneous algorithms according to four different bandwidth scenarios. Increasing the bandwidth from 10 to 40 as it shows is reducing the failure probability of all the algorithms, which is caused by sending more update messages from each controller to the orchestrator. The results show MLNOU algorithm is closest to the optimal in all four scenarios. The Homogeneous algorithm achieves the lowest performance. The evaluation shows MLNOU achieves a failure probability that is 7% less than Homogeneous and 4% less than MLN-SG.

As shown in Fig. 3.6, We increase the number of nodes and links to test the scalability of the algorithm in terms of network size. By increasing the number of links and nodes of a layer in an MLN, the state change rate and message cost of that layer increase. For instance, if we add

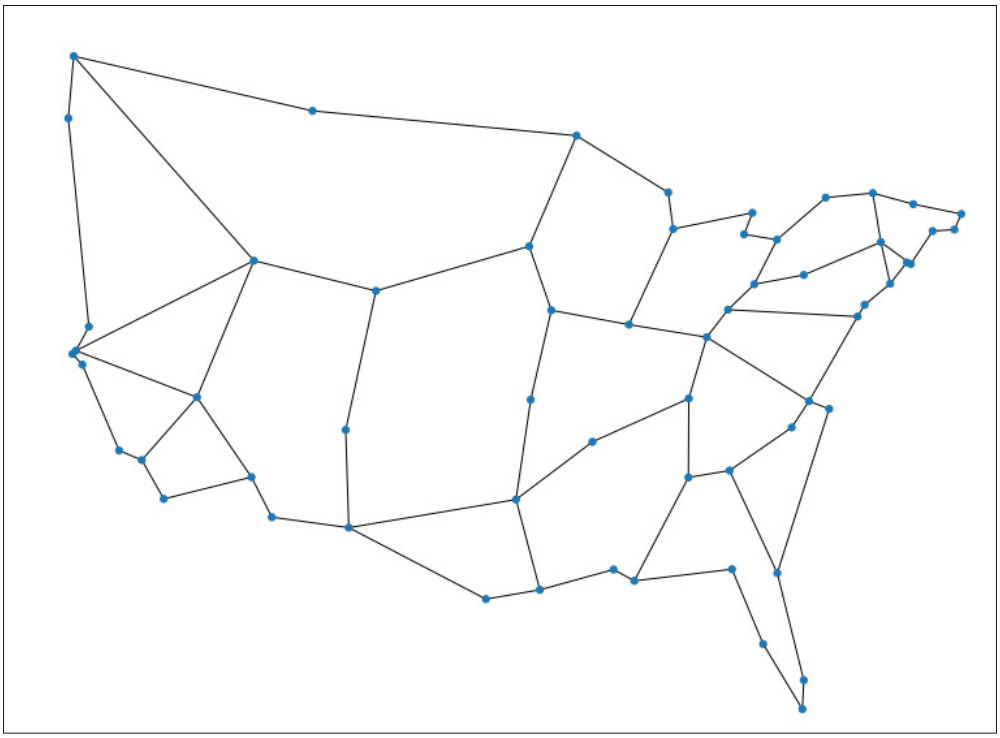


Figure 3.3 Coronet network [Von Lehmen *et al.* (2015)]

10 nodes to a layer and each node sends 1000 reports to its controller every second, in which each report cost is 1 Kbps, the controller message cost to the orchestrator will be increased 1Mbps. Besides, a higher state change rate causes a higher failure probability in the network. To overcome this, by increasing the orchestrator's bandwidth, controllers can update the TS earlier which reduces inconsistency between the orchestrator and the controllers. The runtime increases along with the bandwidth. However, Fig. 3.6 shows the runtime of the optimal algorithm increases much more rapidly than others, which shows the other algorithms are more scalable.

In addition, we investigate the impact of state change rate (λ) on the failure probability, as depicted in Fig. 3.7, by increasing the state change rate, the failure probability increases for all algorithms. Optimal has the lowest probability however, MLNOU achieves the closest results to the optimal, and Homogeneous is the worst. The failure probability achieved by MLNOU is 13.4% less than Homogeneous, and 5.1% less than MLN-SG.

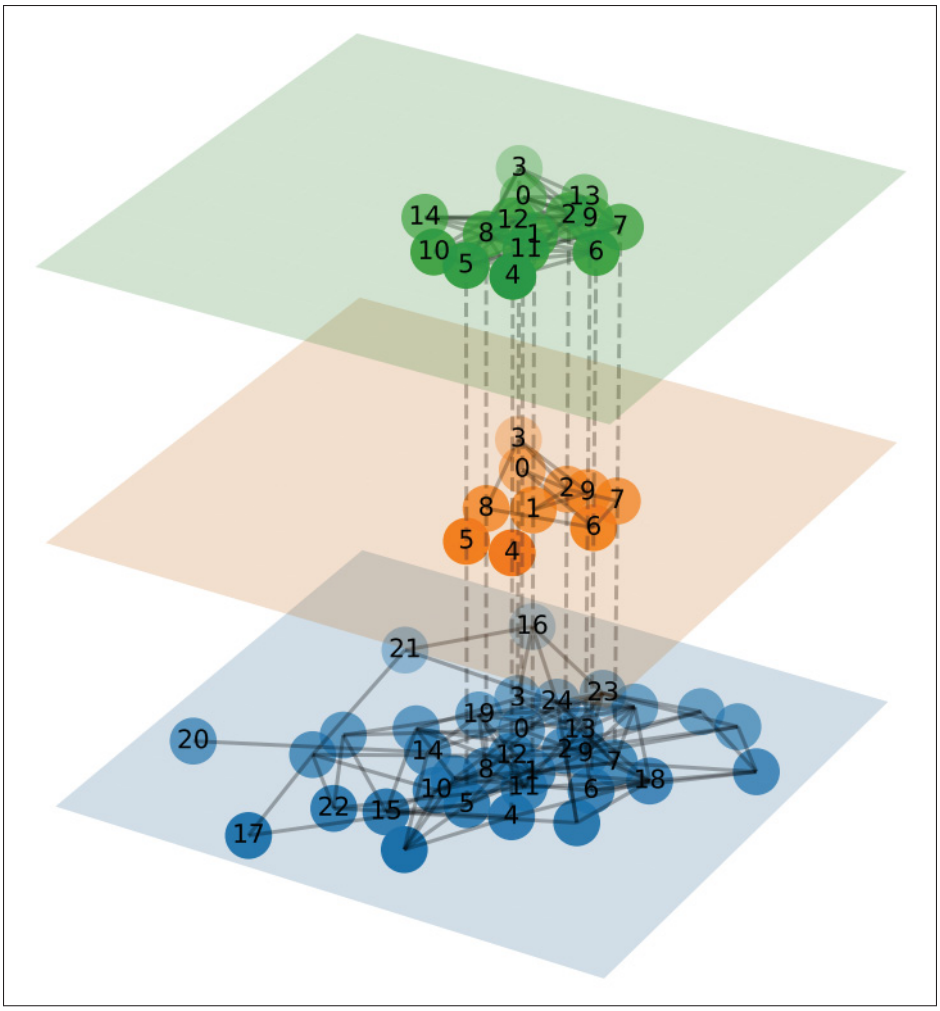


Figure 3.4 Layered Coronet network

3.0.4 Shortest Path Routing

In this section, we describe the experiment settings that we use to evaluate the performance of the shortest path routing application. This application routes packets from their source to their destination through the path of minimum hop count. We use Disjkstra's algorithm [Waleed, Faizan, Iqbal & Anis (2017)] to find the shortest path between a random source and a random destination in our MLN.

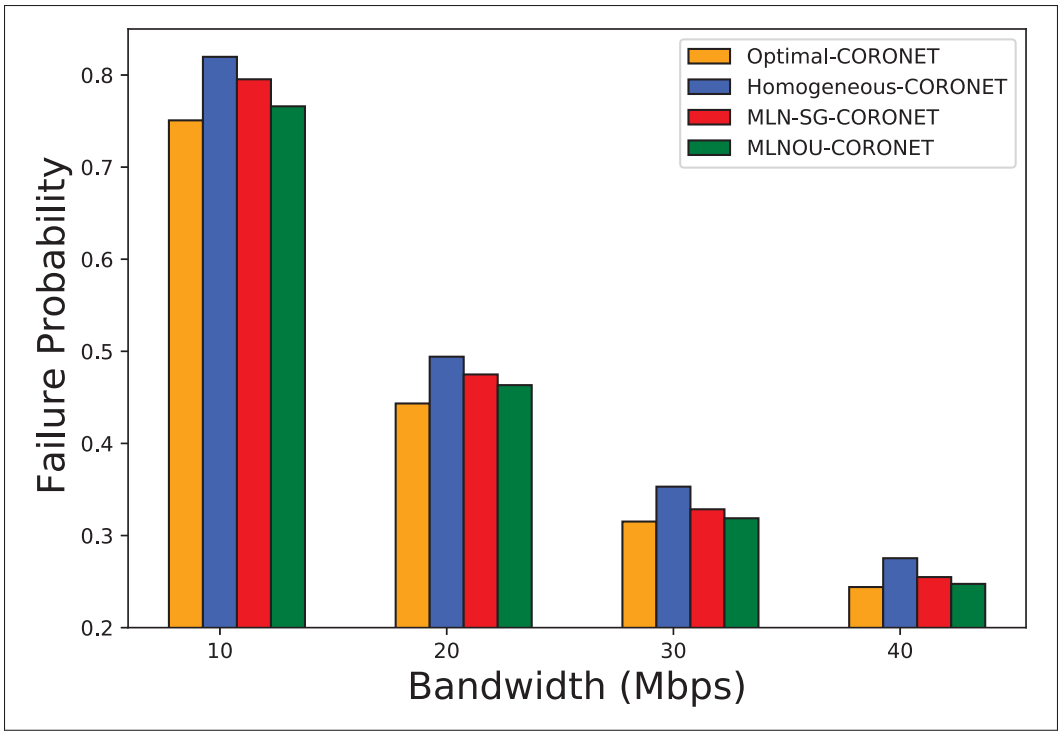


Figure 3.5 Failure Probability comparison

3.0.4.1 Experiment settings

In this experiment, we use the Coronet network with the same topology as in Fig. 3.4. We randomly generate one source and one destination in the IP layer and calculate the shortest path between them by using Dijkstra's algorithm to route the traffic from the source to the destination through multiple layers. We assume links fail or recover randomly every second, and physical link failure in the underlying layers propagates to the virtual links in the upper layers based on a mapping function. There are some studies that investigated the probability of link failure in different layers, for instance [Lee, Lee & Modiano (2011)] studied network reliability in layered networks, where random link failures occur in the lower layer. In such networks, a single failure in the lower layer can result in multiple failures in the upper layer. They extend the traditional polynomial expression for network reliability to the multilayer context and propose approximation algorithms for the failure polynomial using random sampling techniques that operate in polynomial time. Their results show that the probability of a physical link failure in

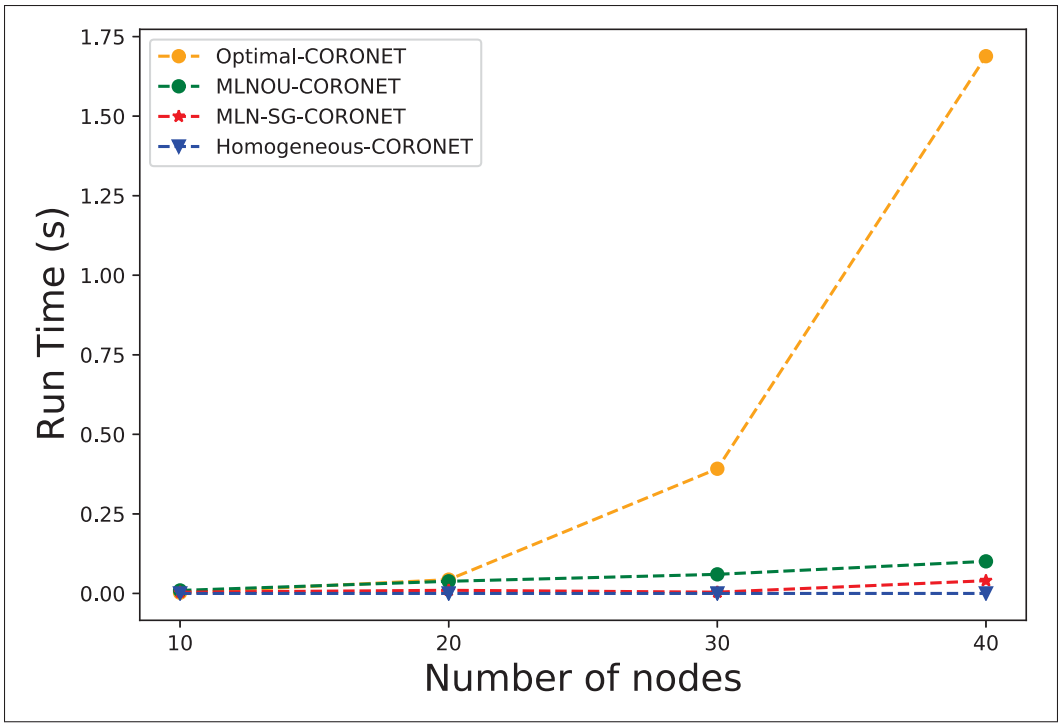


Figure 3.6 Run Time comparison

the DWDM and the OTN layer is very small however, physical link failure in the IP layer occurs more frequently. Therefore, we assume the link failure probability in both the DWDM and OTN layers is 0.001, and this probability in the IP layer is 0.005.

3.0.4.2 Experiment Results

We evaluate the performance of shortest path routing, through the percentage of packets that successfully reach the destination. We implemented this experiment by modeling our network in Python and generating source and destination to route the traffic and fail a link randomly. Then, by using Dijkstra's algorithm we found the shortest path between the source and destination, if the failed link belongs to the paths packets will not reach the destination otherwise they will reach the destination. To generate a better visualization of the effect of updating rate on the orchestrator's performance, we assume that all the controllers update the TS at the same rate.

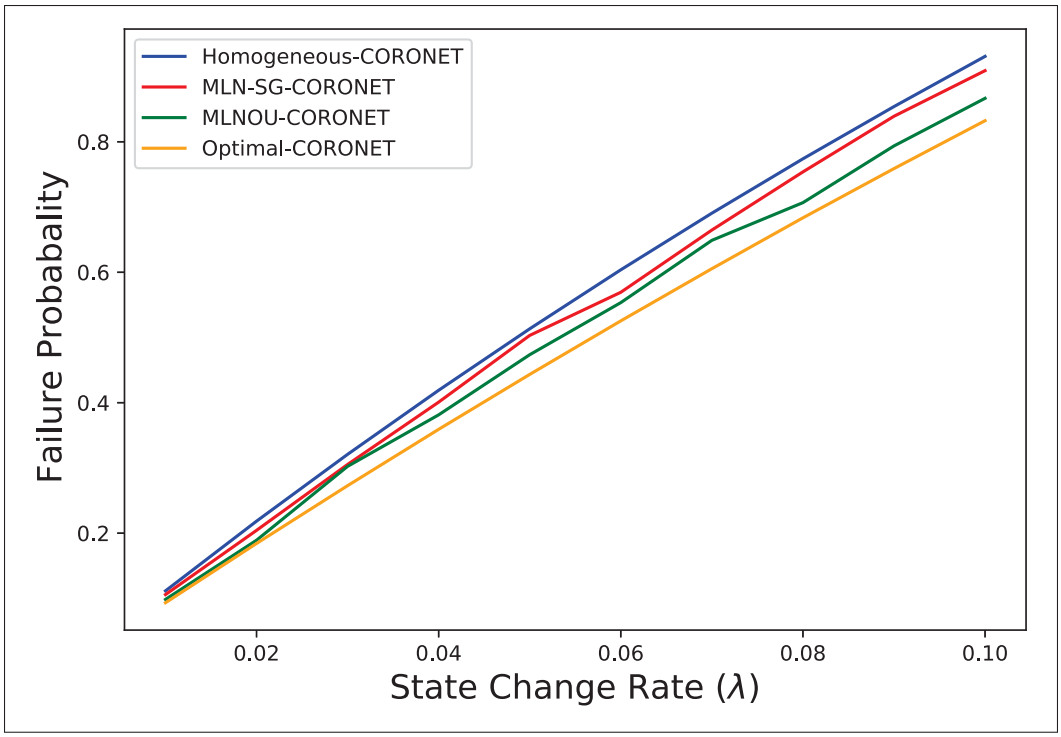


Figure 3.7 State Change Rate comparison

Through this experiment, we show how the updating rate from controllers to the orchestrator affects the orchestrator's performance.

We emulate the performance of the routing algorithm for four different scenarios where all the layers update the orchestrator every 8, 16, 32, and 64 seconds, which means they send 0.125, 0.062, 0.031, and 0.015 messages per second to the orchestrator. We only chose these updating rates to show the correlation between more updating rates and the orchestrator's performance. To avoid randomness, we repeat the experiment for 10 minutes for each of the updating rates. 10 minutes is big enough interval of time to eliminate the effect of randomness. Through these 10 minutes, we choose a random source and destination and route the traffic from the source to the destination, and update the orchestrator based on the specified updating rate, then we calculate the performance of the orchestrator by the percentage of packets that reached the destination. The simulation results that are depicted in Fig. 3.8 show the percentage of successfully routed packets in different updating rate scenarios. As shown, by sending more

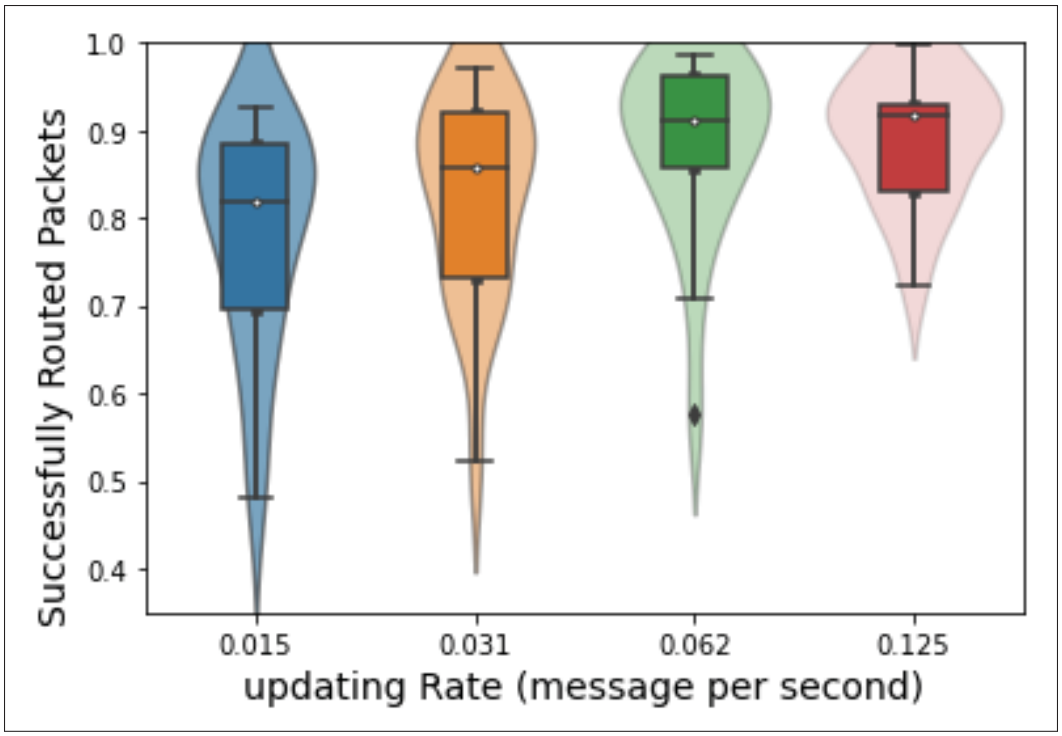


Figure 3.8 Updating rate effect on the normalized value of successfully routed packets

messages per second, the percentage of successfully routed packets increases. For example, by increasing the updating rate from 0.015 to 0.125, the percentage of successfully routed packets increases 10%. In addition, a higher updating rate, results in a decrease in the interquartile range, and in a wider region near the median, which means the distribution of successfully routed packets is more frequent in that region. In Fig. 3.9, we calculate the average number of successfully routed packets and failed packets in different updating rate scenarios. Moreover, by increasing the updating rate from 0.015 to 0.125 the percentage of successfully routed packets increases 9.8%, and the percentage of failed packets decreases. Therefore, the orchestrator has a better performance by receiving a higher number of messages.

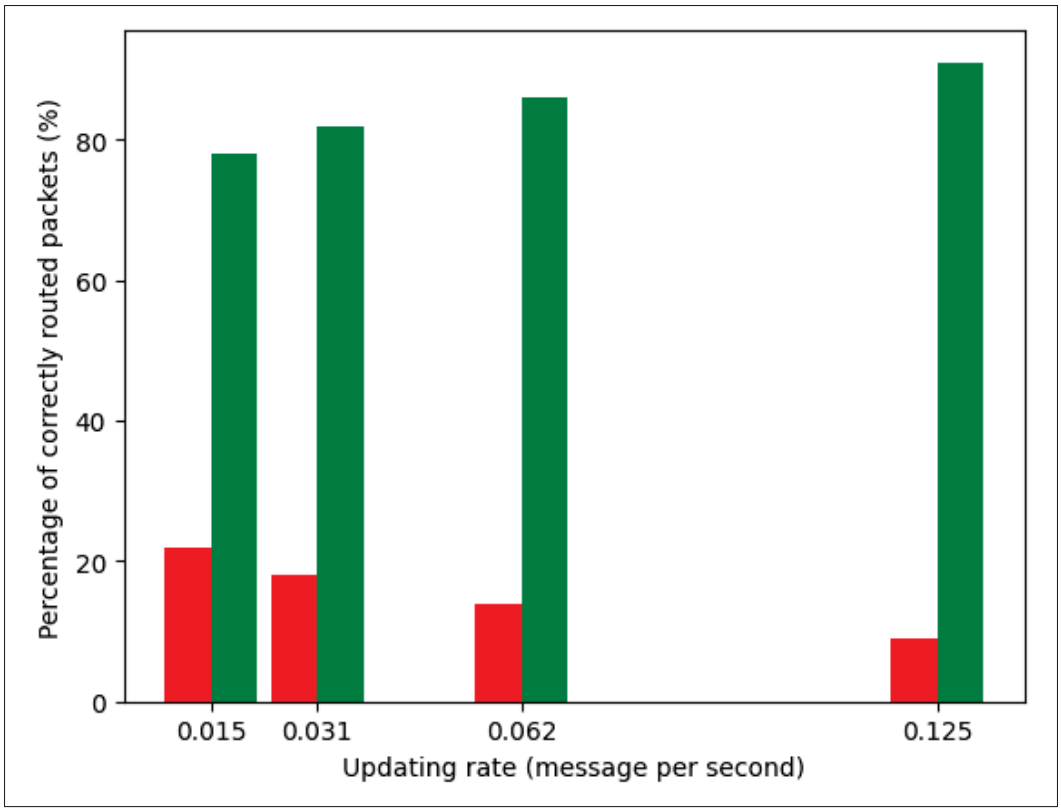


Figure 3.9 Updating rate effect on the successfully routed and failed packets

CHAPTER 4

CONCLUSION AND RECOMMENDATIONS

We can conclude this thesis by stating that we achieved all the objectives that we mentioned in the Introduction of the thesis. In this thesis, we have presented the problem of how often to update the orchestrator in an MLN to reduce the inconsistency in the network, taking into account the bandwidth limitation, the order of updates, and failure propagation time in the network. Our proposed algorithm finds the solution for this non-linear integer optimization problem to minimize the inconsistency between the orchestrator and the controllers in a timely manner. The simulation results show our method outperforms the baselines and approaches the optimal solution, and it is scalable.

We briefly explain how we met the sub-objectives that we mentioned earlier:

- We propose a new updating solution that controllers update the orchestrator in a stateless MLN. Our approach deals with the failure propagation and network limitations such as bandwidth and each controller's maximum message rate (SO1).
- We model the system and formulated it as a non-linear integer optimization problem to minimize the inconsistency between the controllers and the orchestrator while taking into account the updating order, network limitation, and failure propagation (SO2).
- We propose MLNOU algorithm, which is based on the simulated annealing algorithm to solve this NP-hard problem in near real-time. Simulated results show that our proposed solution outperforms the baselines (SO3).

In the future, each layer will include distributed domains that are connected to the layer's controller to increase the complexity of the problem. In addition, in this problem, we assume that failure occurrence probability is based on Poisson distribution, however, in the future, we can gather data from the network, extract the failure distribution out of the data, and define the failure probability based on this distribution instead of using the Poisson distribution.

APPENDIX I

ARTICLE PUBLISHED IN CONFERENCES

This thesis is related to the following paper:

- Optimized Synchronization of the Orchestrator In Hierarchical Multi-layer Networks accepted in IEEE ICC 2023 and will be published in May 2023 [Tirehkar *et al.* (2023)].

BIBLIOGRAPHY

Aguado, A., López, V., Marhuenda, J., de Dios, Ó. G. & Fernández-Palacios, J. P. (2015). ABNO: A feasible SDN approach for multivendor IP and optical networks. *Journal of Optical Communications and Networking*, 7(2), A356–A362.

Alzahrani, S. A. & Katib, I. A. (2018). Impact of varying IP/MPLS capacity module's size in three-layer networks. *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 959–964.

Baranda, J., Mangues-Bafalluy, J., Pascual, I., Nunez-Martinez, J., De La Cruz, J. L., Casellas, R., Vilalta, R., Salvat, J. X. & Turyagyenda, C. (2018). Orchestration of end-to-end network services in the 5G-crosshaul multi-domain multi-technology transport network. *IEEE Communications Magazine*, 56(7), 184–191.

Bjorklund, M. (2016). *The YANG 1.1 data modeling language*.

Botelho, F., Ribeiro, T. A., Ferreira, P., Ramos, F. M. & Bessani, A. (2016). Design and implementation of a consistent data store for a distributed SDN control plane. *2016 12th European Dependable Computing Conference (EDCC)*, pp. 169–180.

Casellas, R., Muñoz, R., Martínez, R., Vilalta, R., Liu, L., Tsuritani, T., Morita, I., López, V., de Dios, O. G. & Fernández-Palacios, J. P. (2015). SDN orchestration of OpenFlow and GMPLS flexi-grid networks with a stateful hierarchical PCE. *Journal of Optical Communications and Networking*, 7(1), A106–A117.

Casellas, R., Martínez, R., Vilalta, R. & Muñoz, R. (2018). Control, management, and orchestration of optical networks: evolution, trends, and challenges. *Journal of Lightwave Technology*, 36(7), 1390–1402.

Chattopadhyay, S., Dai, H. et al. (2019). Maximization of robustness of interdependent networks under budget constraints. *IEEE Transactions on Network Science and Engineering*, 7(3), 1441–1452.

Diamond, S. & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83), 1–5.

Enns, R., Bjorklund, M., Schoenwaelder, J. & Bierman, A. (2011). *Network configuration protocol (NETCONF)*.

Foerster, K.-T., Schmid, S. & Vissicchio, S. (2018). Survey of consistent software-defined network updates. *IEEE Communications Surveys & Tutorials*, 21(2), 1435–1461.

Gill, P., Jain, N. & Nagappan, N. (2011). Understanding network failures in data centers: measurement, analysis, and implications. *Proceedings of the ACM SIGCOMM 2011 Conference*, pp. 350–361.

Gossels, J., Choudhury, G. & Rexford, J. (2019). Robust Network Design for Software-Defined IP/Optical Backbones. *arXiv preprint arXiv:1904.06574*.

Guo, Z., Su, M., Xu, Y., Duan, Z., Wang, L., Hui, S. & Chao, H. J. (2014). Improving the performance of load balancing in software-defined networks through load variance-based synchronization. *Computer Networks*, 68, 95–109.

Huang, D., Cui, M., Zhang, G., Chu, X. & Lin, F. (2020). Trajectory optimization and resource allocation for UAV base stations under in-band backhaul constraint. *EURASIP Journal on Wireless Communications and Networking*, 2020(1), 1–17.

Kantor, M., Biernacka, E., Boryło, P., Domżał, J., Jurkiewicz, P., Stypiński, M. & Wójcik, R. (2019). A survey on multi-layer IP and optical Software-Defined Networks. *Computer Networks*, 162, 106844.

Katta, N., Zhang, H., Freedman, M. & Rexford, J. (2015). Ravana: Controller fault-tolerance in software-defined networking. *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research*, pp. 1–12.

Koulougli, D., Nguyen, K. K. & Cheriet, M. (2020). Hierarchical path computation with flexible ethernet in multi-layer multi-domain networks. *2020 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S. & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), 14–76.

Le, G., Ferdousi, S., Marotta, A., Xu, S., Hirota, Y., Awaji, Y., Tornatore, M. & Mukherjee, B. (2020). Survivable virtual network mapping with content connectivity against multiple link failures in optical metro networks. *Journal of Optical Communications and Networking*, 12(11), 301–311.

Lee, K., Lee, H.-W. & Modiano, E. (2011). Reliability in layered networks with random link failures. *IEEE/ACM Transactions on Networking*, 19(6), 1835–1848.

Levin, D., Wundsam, A., Heller, B., Handigol, N. & Feldmann, A. (2012). Logically centralized? State distribution trade-offs in software defined networks. *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 1–6.

Lin, S.-C., Wang, P., Akyildiz, I. F. & Luo, M. (2018). Towards optimal network planning for software-defined networks. *IEEE Transactions on Mobile Computing*, 17(12), 2953–2967.

Liu, S., Lu, W. & Zhu, Z. (2018). On the cross-layer orchestration to address IP router outages with cost-efficient multilayer restoration in IP-over-EONs. *Journal of Optical Communications and Networking*, 10(1), A122–A132.

Manzanares-Lopez, P., Muñoz-Gea, J. P., Malgosa-Sanahuja, J. & Flores-de la Cruz, A. (2019). A virtualized infrastructure to offer network mapping functionality in SDN networks. *International Journal of Communication Systems*, 32(10), e3961.

Mayoral, A., Vilalta, R., Munoz, R., Casellas, R. & Martánez, R. (2015). Performance analysis of SDN orchestration in the cloud computing platform and transport network of the ADRENALINE testbed. *2015 17th International Conference on Transparent Optical Networks (ICTON)*, pp. 1–4.

Mayoral, A., Vilalta, R., Muñoz, R., Casellas, R. & Martínez, R. (2017). SDN orchestration architectures and their integration with cloud computing applications. *Optical Switching and Networking*, 26, 2–13.

Mayoral, A., Lopez, V., Fernández-Palacios, J. P., Yufeng, Y., Lifen, Z., Wenjun, H., Mingfeng, Z. & Changlong, Y. (2020). First demonstration of YANG push notifications in Open Terminals. *2020 International Conference on Optical Network Design and Modeling (ONDM)*, pp. 1–3.

Mirkhanzadeh, B., Shakeri, A., Shao, C., Razo, M., Tacca, M., Galimberti, G. M., Martinelli, G., Cardani, M. & Fumagalli, A. (2018). An SDN-enabled multi-layer protection and restoration mechanism. *Optical Switching and Networking*, 30, 23–32.

Muñoz, R., Vilalta, R., Casellas, R. & Martínez, R. (2015a). SDN orchestration and virtualization of heterogeneous multi-domain and multi-layer transport networks: The STRAUSS approach. *2015 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pp. 142–146.

Muñoz, R., Vilalta, R., Casellas, R., Martínez, R., Francois, F., Channegowda, M., Hammad, A., Peng, S., Nejabati, R., Simeonidou, D. et al. (2015b). Transport network orchestration for end-to-end multilayer provisioning across heterogeneous SDN/OpenFlow and GMPLS/PCE control domains. *Journal of Lightwave Technology*, 33(8), 1540–1548.

Muqaddas, A. S., Giaccone, P., Bianco, A. & Maier, G. (2017). Inter-controller traffic to support consistency in ONOS clusters. *IEEE Transactions on Network and Service Management*, 14(4), 1018–1031.

Oktian, Y. E., Lee, S., Lee, H. & Lam, J. (2017). Distributed SDN controller system: A survey on design choice. *computer networks*, 121, 100–111.

Poularakis, K., Qin, Q., Ma, L., Kompella, S., Leung, K. K. & Tassiulas, L. (2019). Learning the optimal synchronization rates in distributed SDN control architectures. *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1099–1107.

Qin, Q., Poularakis, K., Iosifidis, G. & Tassiulas, L. (2018). SDN Controller Placement at the Edge: Optimizing Delay and Overheads. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 684-692. doi: 10.1109/INFOCOM.2018.8485963.

Sakic, E., Sardis, F., Guck, J. W. & Kellerer, W. (2017). Towards adaptive state consistency in distributed SDN control plane. *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7.

Salama, M., Ezzeldin, M., El-Dakhakhni, W. & Tait, M. (2022). Temporal networks: A review and opportunities for infrastructure simulation. *Sustainable and resilient infrastructure*, 7(1), 40–55.

Savi, M. & Siracusa, D. (2018). Application-aware service provisioning and restoration in SDN-based multi-layer transport networks. *Optical Switching and Networking*, 30, 71–84.

Srinivasan, S. M., Truong-Huu, T. & Gurusamy, M. (2018). TE-based machine learning techniques for link fault localization in complex networks. *2018 IEEE 6th International conference on future internet of things and cloud (FiCloud)*, pp. 25–32.

Szyrkowiec, T., Santuari, M., Chamania, M., Siracusa, D., Autenrieth, A., Lopez, V., Cho, J. & Kellerer, W. (2018). Automatic intent-based secure service creation through a multilayer SDN network orchestration. *Journal of Optical Communications and Networking*, 10(4), 289–297.

Tirehkar, A., Nguyen, K. K. & Cheriet, M. (2023). Optimized Synchronization of the Orchestrator In Hierarchical Multi-layer Networks. *2023 IEEE International Conference on Communications (ICC)*, pp. 1–6.

Tsai, P.-W., Tsai, C.-W., Hsu, C.-W. & Yang, C.-S. (2018). Network monitoring in software-defined networking: A review. *IEEE Systems Journal*, 12(4), 3958–3969.

Von Lehmen, A., Doverspike, R., Clapp, G., Freimuth, D. M., Gannett, J., Kolarov, A., Kobrinski, H., Makaya, C., Mavrogiorgis, E., Pastor, J. et al. (2015). CORONET: Testbeds, demonstration, and lessons learned. *Journal of Optical Communications and Networking*, 7(3), A447–A458.

Waleed, S., Faizan, M., Iqbal, M. & Anis, M. I. (2017). Demonstration of single link failure recovery using Bellman Ford and Dijikstra algorithm in SDN. *2017 International Conference on Innovations in Electrical Engineering and Computational Technologies (ICIEECT)*, pp. 1–4.

Wazirali, R., Ahmad, R. & Alhiyari, S. (2021). SDN-openflow topology discovery: an overview of performance issues. *Applied Sciences*, 11(15), 6999.

Williams, M. J. & Musolesi, M. (2016). Spatio-temporal networks: reachability, centrality and robustness. *Royal Society open science*, 3(6), 160196.

Yan, S., Aguado, A., Ou, Y., Wang, R., Nejabati, R. & Simeonidou, D. (2017). Multilayer network analytics with SDN-based monitoring framework. *Journal of Optical Communications and Networking*, 9(2), A271–A279.

Yu, T., Hong, Y., Cui, H. & Jiang, H. (2018). A survey of Multi-controllers Consistency on SDN. *2018 4th International Conference on Universal Village (UV)*, pp. 1–6.

Zhang, B., Wang, X. & Huang, M. (2018). Adaptive consistency strategy of multiple controllers in SDN. *IEEE Access*, 6, 78640–78649.

Zhang, M., Wang, X., Jin, L. & Song, M. (2021a). Cascade phenomenon in multilayer networks with dependence groups and hierarchical structure. *Physica A: Statistical Mechanics and its Applications*, 581, 126201.

Zhang, X., Cui, L., Wei, K., Tso, F. P., Ji, Y. & Jia, W. (2021b). A survey on stateful data plane in software defined networks. *Computer Networks*, 184, 107597.

Zhang, Z., Ma, L., Poularakis, K., Leung, K. K., Tucker, J. & Swami, A. (2019). Macs: Deep reinforcement learning based sdn controller synchronization policy design. *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pp. 1–11.

Zhu, L., Karim, M. M., Sharif, K., Xu, C., Li, F., Du, X. & Guizani, M. (2020). SDN controllers: A comprehensive analysis and performance evaluation study. *ACM Computing Surveys (CSUR)*, 53(6), 1–40.