

AMERICAN UNIVERSITY OF BEIRUT

On Centralized and Distributed Control of
Communication Networks

by

SARAH ANIS ABDALLAH

A dissertation
submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
to the Department of Electrical and Computer Engineering
of the Maroun Semaan Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
February 2020

AMERICAN UNIVERSITY OF BEIRUT

On Centralized and Distributed Control of Communication Networks

by

SARAH ANIS ABDALLAH

Approved by:



Dr. Ali Chehab, Professor

Chairman

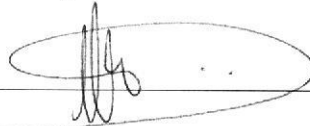
Electrical and Computer Engineering



Dr. Ayman Kayssi, Professor

Advisor

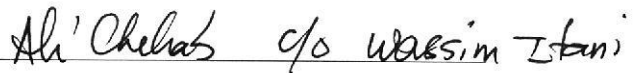
Electrical and Computer Engineering



Dr. Imad Elhajj, Professor

Member of Committee

Electrical and Computer Engineering



Dr. Wassim Itani, Associate Professor

Member of Committee

Electrical and Computer Engineering, Beirut Arab University

Dr. George Sakr, Assistant Professor

Member of Committee

Electrical and Computer Engineering, St Joseph University



Date of dissertation defense: January 17, 2020

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: ABDALLAH SARAH ANIS
Last First Middle

Master's Thesis Master's Project Doctoral Dissertation

I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after: **One year from the date of submission of my dissertation.**
Two ___ years from the date of submission of my dissertation.
Three ___ years from the date of submission of my dissertation.

 19/2/2020
Signature Date

This form is signed when submitting the thesis, dissertation, or project to the University Libraries

Acknowledgements

My recognition and gratitude are addressed to my committee members, Dr. Ayman Kayssi, Dr. Ali Chehab, Dr. Imad Elhajj, Dr. Wassim Itani, and Dr. Georges Sakr for their guidance and advice throughout this work. I would like to thank my advisors, Dr. Ayman Kayssi, Dr. Ali Chehab, and Dr. Imad Elhajj, for the support they provided me during my PhD studies. I would also like to thank my research group who made the graduate studies comfortable and enjoyable. The deepest gratitude go to my parents who taught me how to fight and never give up, and who provided me with the best support. Special thanks are for my friends, for their great help and their continuous support.

An Abstract of the Dissertation of

Sarah Anis Abdallah for Doctor of Philosophy
Major: Electrical and Computer Engineering

Title: On Centralized and Distributed Control of Communication Networks

Networks are facing many challenges with the exponential increase in devices and traffic. The lack of seamless scalability, programmability, and remote management of traditional networks requires the investigation of new networking paradigms. Software-Defined Networking (SDN), with centralized control, promises to offer the above three features in addition to other advantages. However, legacy networking protocols have been the subject of research for a long time, to the extent that some of the techniques they use and the advantages they provide in some circumstances should not be replaced. At the same time, SDN, like any other technology, presents some shortcomings, one of which is the single point of failure.

On the other hand, the centralization or distributedness of control exists beyond the area of telecommunication. Some systems, such as the nervous system or administrative and political regimes, exhibit the same interrelationship between the two types of control. These systems serve as an inspiration to designing hybrid control planes in data networks. This thesis aims to model and compare the performance of both types of control in communication network using OSPF and Openflow, identifying the strengths and weaknesses of each. It then aims to draw the control rules that govern the inter-operability of centralized and decentralized control from the comparative study and in the areas defined above, and use them to model an adaptive hybrid control system that exploits the advantages of both centralized and distributed control, with application to networks. The system is then allowed to shift either completely or partially from a centralized system to a distributed environment so as to deliver a desired performance. The model is then applied to telecommunication networks.

Much work in the literature concerns operating and managing networks that contain legacy and SDN nodes. Hybrid SDN switches that support both SDN and

legacy routing protocols have also been presented and studied. Our technique differs from the literature because we study and control changing network technologies at the granularity of the switch. We combine notions of control systems, SDN networks, traditional IP networks and hybrid switches to create optimization systems that monitor the network conditions and tunes each switch to function under SDN or IP accordingly. This technique would not only allow exploiting the advantages of both SDN and legacy networks for enhanced performance, it would also allow us to study the dynamics between centralized and decentralized control instances of any system.

Results showed that the proposed methods deliver up to 62% better performance compared to baseline technologies when network conditions are at 50% of their maximum allowed values, and up to 95% better performance when network conditions are at 100% of their maximum allowed values.

Contents

Acknowledgements	v
Abstract	vi
1 Introduction	1
1.1 Contributions	3
1.2 Dissertation Structure	4
2 Background	6
2.1 Distributed Routing Protocols in Traditional Networks	6
2.2 Introduction to Software Defined Networking (SDN)	9
2.3 Introducing Logical Versus Physical Centrality	11
2.4 Comparative Analysis	12
2.5 Overview of Centralization and Decentralization in Different Areas	14
2.5.1 History and Evolution of Network Control Planes	14
2.5.2 Models of Centralized and Distributed Control	21
3 Effect of Centralized and Distributed Control on Networks	29
3.1 Hybrid Control Planes in Networks	29
3.1.1 Integrating Traditional Routing Protocols with SDN	30
3.1.2 Routing Islands	33

3.1.3	Hybrid Switches	35
3.1.4	Consistency of Control Planes	37
3.2	Effect of Different Types of Control on Network Services	38
3.2.1	Middle-boxes	39
3.2.2	MPLS	49
3.2.3	Multicast	52
3.2.4	VxLAN	55
3.2.5	QoS	57
3.3	Consequences on Network Management	59
3.3.1	NM Layer for SDN	62
3.3.2	Proposed Management Schemes	64
3.3.3	Guidelines for a Management Framework	68
3.3.4	POC: System Model and Implementation	71
4	Performance Modeling and Testing of Centralisation (SDN) Versus Decentralisation (OSPF)	78
4.1	Network Convergence in SDN Versus OSPF Networks	78
4.1.1	Network convergence	79
4.1.2	Theoretical Model	83
4.1.3	Testing and Results	87
4.1.4	Simulation and Results	90
4.2	Performance Analysis of SDN vs. OSPF in Diverse Network Environments	97
4.2.1	Extension to the Previous Convergence Model	97
4.2.2	Extended Convergence Model	100
4.2.3	Implementation and Testing	104

4.2.4	Discussion and Analysis	119
5	Model of Adaptive Control in Hybrid Systems	123
5.1	Offline Recommender System	124
5.1.1	Fuzzy Decision System Design	124
5.1.2	Defining the Inference Rules	127
5.1.3	Financial Implications of the System Decisions	129
5.1.4	Implementation and Case Studies	133
5.2	Modeling the General Optimization System	139
5.2.1	HCA Cost	141
5.2.2	SH Cost	142
5.2.3	Uncertainty Cost	142
5.2.4	SS Cost	142
5.3	Application to Telecom Networks:	143
5.3.1	Adaptive Hybrid System (AHS): Overview and Design	143
5.4	Preliminary Optimization Design and Results	147
5.4.1	Optimization Problem Formulation	149
5.4.2	Modeling of Input Parameters	156
5.4.3	Results	160
5.5	Final Model Formulation	165
5.5.1	DO problem Formulation	168
5.5.2	GO Problem Formulation	178
5.5.3	Design Alternatives	179
5.6	Results and Discussion	181
5.6.1	DO Testing	181
5.6.2	GO Testing	190

6	Implementation and Results of Final Complete System	195
6.1	Implementation Details	195
6.1.1	The Hybrid Switch (HS):	196
6.1.2	The HS State Control System – the Rule Installer Block:	198
6.1.3	The Monitoring System:	200
6.1.4	The Optimization System/Decision Making:	201
6.1.5	SDN Controller	201
6.1.6	Experiment Overview	202
6.2	Testing and Results	203
6.2.1	Scenario 1: Effect of Communication Delay with Controller	205
6.2.2	Scenario 2: Effect of Topology Change Rate	209
6.2.3	Scenario 3: Effect of Rate of Unknown Packets	212
6.2.4	Scenario 4: Effect of Local Delay	215
7	Summary of Results and Analysis	222
7.1	Overall Results and Analysis	222
8	Conclusions and Future Directions	226
8.1	Conclusion Related to Network Services and Management	227
8.2	Conclusion Related to Modeling Network Convergence	227
8.3	Conclusion Related to the Proposed Designs and the Adaptive Optimization System	228
8.4	Future Directions	229
	Bibliography	233

List of Figures

2.1	SDN (bottom) vs. conventional networking (top)	10
2.2	SDN layers as defined by ONF	10
2.3	Taxonomy of control planes	11
2.4	Evolution of the control planes; red: centralized control, orange: hierarchical routing, green: distributed control, blue: hybrid control	17
2.5	Allocation of control tasks	28
3.1	MPLS label	50
3.2	Map of management functions for building management schemes for SDN	70
3.3	Proposed management framework based on management functions categorization	72
3.4	Comparison of the normalized performance cost of the 3 scenarios	74
3.5	Performance cost with varying controller capacity	76
3.6	Performance cost with varying controller capacity in semi-log scale	76
3.7	Performance cost with varying Delay	77
4.1	Simulated network of 50 nodes	84
4.2	Effect of link delay on convergence delay	88
4.3	Effect of fault location on convergence delay	89

4.4	OSPF convergence	91
4.5	SDN convergence	91
4.6	Convergence delay in the OSPF network vs. SDN	94
4.7	Convergence for large network link delays	96
4.8	Datacenter layered architecture and distances	105
4.9	Wan architecture (background image taken from maps.google.com)	105
4.10	Fault location effect on convergence in datacenter	109
4.11	Fault location effect on convergence in WAN	109
4.12	Network congestion effect on convergence in datacenter	110
4.13	Network congestion effect on convergence in WAN	110
4.14	Controller load effect on convergence in datacenter	112
4.15	Controller load effect on convergence in WAN	112
4.16	Controller capacity effect on convergence in datacenter	113
4.17	Controller capacity effect on convergence in WAN	113
4.18	Control link BW effect on convergence in datacenter in log-log scale	113
4.19	Control link BW effect on convergence in datacenter in log-log scale	114
4.20	Controller capacity effect for different network congestion values in the datacenter	114
4.21	Control capacity effect for different network congestion values in WAN	115
4.22	Effect of in-band and out-of-band controller on convergence speed in the datacenter	116
4.23	Effect of in-band and out-of-band controller on convergence speed in the WAN	116
4.24	Convergence delays of SDN and OSPF in the WAN	118
4.25	Effect of the control plane delay on convergence in SDWAN . . .	118

5.1	Case study 1	134
5.2	Case study 2	135
5.3	Progressive scenario: initial state	136
5.4	Progressive scenario: round 2	137
5.5	Progressive scenario: change in update frequency	138
5.6	OS emplacement within system environment	140
5.7	Optimization general model - GO: global optimization, DO: distributed optimization	140
5.8	The hybrid network (HN)	145
5.9	OS functional block diagram	146
5.10	AHS sub-systems and components	147
5.11	Preliminary design high-Level model	148
5.12	Performance of our system in comparison with fully centralized and fully distributed settings	165
5.13	Min delay to controller - max rate of topology change	166
5.14	Max delay to controller - min rate of topology change	167
5.15	Min delay to controller - min rate of unknown packets	167
5.16	Min delay to controller - max rate of unknown packets	168
5.17	Min rate of topology change - min rate of unknown packets	169
5.18	Max rate of topology change - max rate of unknown packets	169
5.19	Min delay to controller – min network size	170
5.20	Max delay to controller – max network size	170
5.21	Optimization solution for different network conditions	171
5.22	Effect of the condition on the optimization results - a	171
5.23	Effect of the condition on the optimization results - b	172
5.24	Control plane states for DO with t_1 in random conditions	182

5.25	Control plane states for DO with t_2 in random conditions	182
5.26	Percentage of HS's having distributed control wrt. time	183
5.27	Total percentage of HS's having distributed control over the total time period	184
5.28	Control plane states for fully centralized network conditions . . .	185
5.29	Percentage of HS's having distributed control wrt. time	186
5.30	Total percentage of HS's having distributed control over the total time period	187
5.31	Control plane states for fully distributed network conditions . . .	187
5.32	Percentage of HS's having distributed control wrt. time	188
5.33	Total percentage of HS's having distributed control over the total time period	189
5.34	Control plane states for DO with t_1 in incremental conditions . .	189
5.35	Control plane states for DO with t_2 in incremental conditions . .	190
5.36	Percentage of distributed control vs. time	191
5.37	Total percentage of distributed control	191
5.38	Control plane states for standalone GO in incremental conditions	192
5.39	Percentage of distributed control vs. time	193
5.40	Control plane states for our proposed optimization in incremental conditions	194
6.1	Implementation design diagram	196
6.2	Table 0 rules for OSPF mode	199
6.3	Table 1 rules for SDN mode	200
6.4	Testing workflow	204
6.5	Network architecture	205

6.6	HS states for DO with threshold t_1	206
6.7	HS states for DO with threshold t_2	206
6.8	HS states for GO with decision based on results of DO	207
6.9	HS states for GO with decision based on delay values	207
6.10	Latency with respect to delay for the different network designs in semi-log scale	209
6.11	HS states for DO with threshold t_1	210
6.12	HS states for DO with threshold t_2	211
6.13	HS states for GO with decision based on results of DO	211
6.14	HS states for GO with decision based on delay values	212
6.15	Latency with respect to rate of topology change for the different network designs in semi-log scale	213
6.16	Switches states for DO with threshold t_1	213
6.17	Switches states for DO with threshold t_2	214
6.18	Switches states for GO with decision based on results of DO	214
6.19	Switches states for GO with decision based on delay values	215
6.20	Latency with respect to rate of unknown traffic for the different network designs in semi-log scale	216
6.21	Delay between switches and controller	216
6.22	Switches states for DO with threshold t_1	217
6.23	Switches states for DO with threshold t_2	218
6.24	HS states for GO with decision based on maximum delay	218
6.25	HS states for GO with decision based on mean delay	218
6.26	Switches states for GO with decision based on results of DO	218
6.27	Latency for the different network designs in semi-log scale	219

8.1	IOT network	231
8.2	PLC-DCS system	232

List of Tables

- 3.1 Network services in traditional networks, SDN and hybrid networks 60
- 3.2 Proposed FCAPS allocation in SDN 69
- 3.3 Simulation scenarios 74

- 4.1 Variables 85
- 4.2 Simulation parameters for scenario 1 87
- 4.3 Simulation parameters for scenario 2 88
- 4.4 Simulation parameters for scenario 3 90
- 4.5 Experiment settings 90
- 4.6 Variables and parameters 102
- 4.7 Distances between the WAN sites shown in Figure 4.9 106
- 4.8 Experiments 106

- 5.1 Variables and parameters 156
- 5.2 Input modeling 160
- 5.3 Variables and parameters 173
- 5.4 Variables and parameters 178

- 6.1 Parameters for scenario 1 205
- 6.2 Parameters for scenario 2 209
- 6.3 Parameters for scenario 3 212

6.4	Percentage improvement of proposed methods compared to base- line SDN at 50% and 100% of maximum delay to controller	220
6.5	Percentage improvement of proposed methods compared to base- line OSPF at 50% and 100% of maximum rate of topology change	221
6.6	Percentage improvement of proposed methods compared to base- line OSPF at 50% and 100% of maximum rate of traffic corruption	221

To my Lily...

Chapter 1

Introduction

Scalability, security, and availability are harder to guarantee with today's networks due to the distributed control across the network elements, the lack of standardization and several other limitations. These characteristics are however crucial in a world where almost everything is connected, especially critical services such as healthcare platforms, connected cars, etc. In fact, and throughout the years, the network technologies have evolved to adapt to the changing requirements of each period and deliver the required performance. As such, SDN emerged recently as an attractive new form of networks, which offers centralized control and easy programmability, and promises to resolve many of the shortcomings faced today [1].

However, hybrid deployment of SDN became a major concern mainly because networks are already deployed everywhere, and shifting to SDN would require updating the whole world. Moreover, some IP protocols are mature and present solid attributes. In fact, traditional practices and distributed protocols have been used, maintained and enhanced throughout the years, and thus provide some characteristics that a relatively new technology such as SDN does not yet

possess [2]. These can be used to address shortcomings of the centralized SDN such as the single point of failure.

At the same time, the general question of distributed versus centralized control in large-scale systems is a very interesting one. In this domain, the traditional view in telecommunications was to use centralized control, while the Internet relies on distributed control and seems to have settled the debate for some time. With the advent of Software-Defined Networks, centralized control is challenging the distributed model again. But the answer may not be to go with one approach over the other; the optimal solution lies somewhere in between and depends on several aspects of the system.

On the other hand, the centralization or distributedness of control exists beyond the area of networks or telecommunication; in fact, some systems exhibit the same interrelationship between the two types, such as the nervous system or administrative and political regimes. Such systems can be exploited to provide guidelines on designing hybrid control planes in networks.

Given all the above, this work thus proposes a hybrid approach that dynamically adjusts the level of centralization versus “distributedness” in a system, to deliver optimal performance. The model, although general, is applied to telecommunication network; the network as such would support both SDN and Legacy networking protocols, leveraging the advantages of each technology given the network conditions. This will result in a controlled network that shifts technologies given the current network conditions, in order to deliver the desired performance. The hybrid network can be used as a solution for the coexistence of IP and SDN networks and incremental deployment of SDN. But most importantly, the hybrid network will ensure better performance as it is adaptively tuned between SDN, IP and different combination of both based on its condition. This technique will

allow the full exploitation of the advantages of both SDN and IP protocols, because the network and (or) switch can “decide” when to use SDN and when to revert to IP, depending if SDN is more favorable in its current condition or not. Subsequently, this work is to be extended to other systems that exhibit the same distributed versus centralized behavior.

1.1 Contributions

Our contributions in this dissertation can be summarized as follows:

- Present a study of Hybrid Networks, and their effects on different network services.
- Design a network management scheme for SDN networks that supports the deployment of Hybrid Networks.
- Model of the convergence process of both SDN and OSPF and the related network parameters in a general network, WANs, and datacenters.
- Present a study of systems that exhibit both types of control, namely administrative systems and nervous systems.
- Design and implement a recommender system for network technology choice based on network criteria using fuzzy logic.
- Design an adaptive hybrid optimization system composed of distributed and global optimization systems that can operate as standalone systems but also together in a hybrid form.
- Implement a complete hybrid network simulator with all related modules and the corresponding testing scheme.

1.2 Dissertation Structure

The rest of the dissertation is organized as follows: In Chapter 2, we introduce the different topics that are related to this work. We start by presenting routing protocols, and then introducing SDN, the new form of networks, pointing out the advantages and disadvantage of each. We also define Logical versus physical centrality, defining thus the taxonomy we use throughout this thesis. Then in Chapter 3, we describe hybrid networks (HN) and elaborate on their operation. We also present a literature survey on several of the routing techniques used in the different forms of HNs. We then study the effect of HNs on network services such as MPLS, middleboxes, and so on, and investigate the consequences of HN on network management. In this regard, we propose a management scheme that is able to support the shifting nature of networks. In Chapter 4, we study the dynamics between centralization and de-centralization of control in in three areas: political sciences, nervous system and 5G, and deduce the control patterns that each system uses to shift from its centralized version to its decentralized one and vice-versa. These interaction are later used in chapter 6 to model the decision making system. In chapter 5, we revisit networks, and conduct a theoretical performance modeling for SDN and an IP routing protocol, OSPF, in the aim of proving that each technology does perform better under varying network conditions. The analytical models are then validated against experimental testing. Furthermore, the models are extended to account for two disparate environments, WANs and Datacenters, to show that the underlying infrastructure also affects performance. The work in this section is carried out in order to prove that the two control planes perform differently under different conditions, and thus justify our proposed adaptive hybrid model. In Chapter 6, we present the

different proposals and models for the decision making system, starting from an offline recommender system up to an adaptive hybrid optimization system, which allows switches to dynamically change the state of their control plane given network conditions, along with the implementation and results of each proposal. This is followed by summary of results, analysis, and application of model to other areas in chapter 7. Finally, we present our conclusions and shed light on future directions in Chapter 8.

Chapter 2

Background

Traditional IP networks rely on distributed routing protocols, where each router is not only a forwarding agent but also a decision maker. SDN separated the data and the control planes, and centralized the control within a single entity: the SDN controller.

2.1 Distributed Routing Protocols in Traditional Networks

When a router receives a packet, it checks the destination IP address against its routing table and forwards the packet to the next hop via the corresponding interface. If no entry matches the destination IP, the packet is sent to the default route, if it exists, otherwise it is dropped. Routing algorithms populate the routing tables; they are categorized as Link State routing protocols or Distance-Vector routing protocols. The distance vector routing protocol, exemplified by the Routing Information Protocol (RIP) in small networks, populates routing tables based on the Bellman-Ford algorithm. With this protocol, each node (router)

regularly informs all its neighbors of all accessible nodes through it along with the cost it takes for it to reach them. Collecting information from all neighbors, a node can then identify the best next-hop that leads to a specific destination along with the associated cost, and populates its routing table. A node also replaces any entry to a destination with a new one, if this latter is found to be less costly. Eventually, the algorithm converges and all nodes will know the best hop with the least cost for any destination in the network.

In case of a node failure, all nodes, having the failed one as a next hop, would remove the entry from their tables, advertise the event to neighboring nodes, and find a new best next hop for the corresponding destination. Iteratively, nodes advertise new information and the algorithm re-converges [3]. Link State routing differs from Distance Vector routing in the fact that the topology advertisements are sent to the whole network and not just to neighboring nodes. Link State routing algorithms are characterized by a connectivity map computed at each node (router) of the network. Each router independently builds its map by informing the whole network of the nodes that it can reach. Specifically, the routers flood packets (Link State Packets) that convey information about its direct connectivity to nodes along with the cost of each link. On receipt of this information, each router separately reproduces the complete topology of the network and computes the least-cost connectivity graph to all nodes in the network, employing Dijkstra's or any other shortest path algorithm. The resulting graph is then used to populate the appropriate routing table. Among the Link State routing protocols, we cite intermediate system to intermediate system (IS-IS) and open shortest path first (OSPF) [3].

Routing protocols are also classified with respect to their zone of application: Intra-domain such as RIP and OSPF, and Inter-domain routing protocols

such as BGP. The first type is usually deployed within an autonomous system (AS), whereas the second is used to interconnect different AS's under distinct administrative authorities. BGP is the most prominent and widely deployed inter-domain routing protocol. It is similar to Distance Vector protocols (DVPs) with one special characteristic: unlike other DVPs, which rely on the number of hops, BGP saves and forwards all the AS numbers through which the packet should be routed to reach a destination. Therefore, BGP is often referred to as "Path Vector Routing Protocol". BGP operates in two modes: external BGP (EBGP) and internal BGP (IBGP). EBGP is used to connect 2 BGP peers across different AS's, and IBGP is used when an AS is connected to many other AS's via different border BGP routers [4]. Since BGP connects independent AS's having different administrative rules and conditions, it enforces policy deployment [5].

In IP networks, switches detain a different role from that of a router. Switching occurs at Layer-2 within the same network or subnet. Since switches do not alter the Ethernet frame, their operation is often referred to as "transparent bridging". A switch usually saves a forwarding table built when the switch performs "learning". In fact, when a switch receives a frame on one of its Ethernet ports, it reads both the destination address and the source address. By saving the source address in the table, the switch can know which MAC addresses can be reached and the corresponding ports through which it can do so. When the switch receives a frame for a destination that is not present in the table, it floods the frame on all the ports except the one that it received the packet from [6].

2.2 Introduction to Software Defined Networking (SDN)

SDN is a developing architecture for networks where the data plane (the infrastructure composed of routers and switches) is decoupled from the control plane [7]. In other words, SDN deprives the switches and routers from the traditional function of deciding on the forwarding of packets, and grants it to a central controller which commands the network devices on how to forward packets.

A comparative scheme between SDN and a conventional (inter-network) system is shown in Figure 2.1. Instead of having distributed functions and middle boxes (top scheme), SDN elevates all functions to a top layer called the SDN controller. The Controller is in charge of implementing all functions such as routing, intrusion detection and firewalling, through applications that run on top of it [8].

Figure 2.2 shows the communication among the SDN layers. The data plane contains all network elements and communicates with the control plane via southbound interfaces. The most deployed southbound protocol is OpenFlow, as standardized by the Open Networking Foundation (ONF). The controller resides in the control plane; it is responsible for enforcing rules in the data plane. The control plane exposes northbound interfaces that allow applications sitting on top of the control plane to communicate with the latter. A management layer perpendicular to all three layers is in charge of all management functions [9].

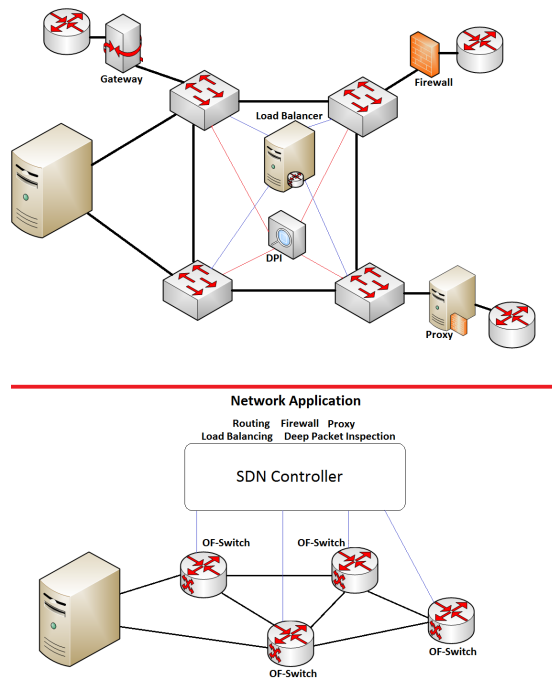


Figure 2.1: SDN (bottom) vs. conventional networking (top)

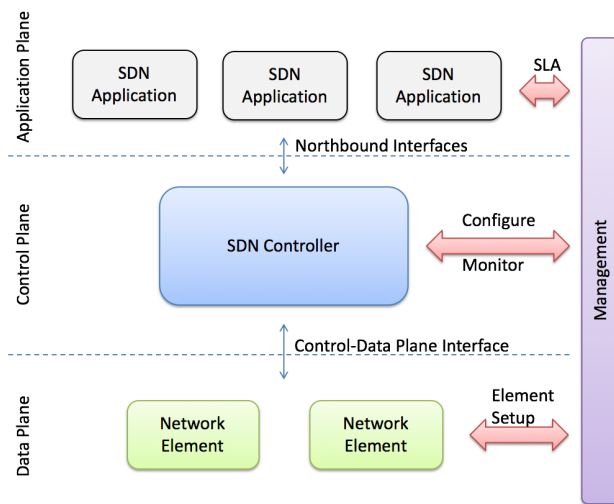


Figure 2.2: SDN layers as defined by ONF

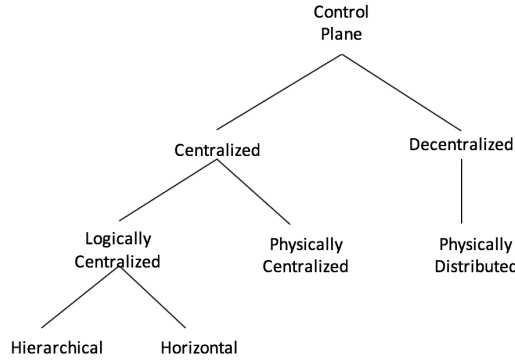


Figure 2.3: Taxonomy of control planes

2.3 Introducing Logical Versus Physical Centrality

Figure 2.3 shows the taxonomy that we adopt in our work in relation to the difference between centralization, decentralization and distribution.

Control planes can be generally centralized or decentralized. Decentralized control planes are physically distributed control nodes that autonomously perform control activities and decisions, such as control planes of traditional IP protocols [10]. Centralized control planes are classified into either physically centralized or logically centralized. In physically centralized systems, the central controller physically resides on one node in the system, the SDN controller. The notion of logical centrality was introduced as a solution to the contested centralized nature of SDN, which makes it non-scalable and prone to failure. Consequently, many multi-level controller propositions emerged [11, 12, 13]. These methods aim at providing scalable SDN networks that can adapt to growing networks, and implement load balancing in order to avoid the threat of a single point of failure. As a result, even though solutions that implement distributed controllers and databases are considered to solve the scalability issues, control is still considered

logically centralized. In logically centralized systems, control plane instances physically reside on different nodes. Logically centralized solutions cover SDN proposals with many physical controllers that cooperate to serve the control plane. Examples of logically centralized control planes include horizontal designs where the controller's instances share a network-wide view. In horizontal deployments, all physical controllers have the same role. In ASIC [11], a logical controller layer implementing a load balancer forwards requests to an underlying layer of physical controllers that are all connected to the global network database. Hyperflow [14] also implements horizontal deployments where all controllers share a view of the whole network, but only control switches in their proximities.

Logically centralized systems also include hierarchical deployments like Kandoo [15] where local controllers having local knowledge report to a root controller having global network knowledge. Kandoo considers hierarchical distributions of root and local controllers, and distribute the control tasks to either control level based on the span and frequency of the corresponding network events.

A special case of local controllers and more drastic proposals such as DIFANE [16] and DevoFlow [17], which utilize the switches to offload the controller by injecting wildcard rules in them. The switches then constitute extensions to the controllers. Even though in this case the control plane is pushed to the extremity of the network, it is still considered as part of the logically centralized system because it still relies on the SDN controller.

2.4 Comparative Analysis

Each of the two types of networks has its own advantages and disadvantages. SDN removes the burden of distributed route calculation at the Network Elements

level, which provides network administrators more control over their systems. For example, SDN allows network administrators to easily create routing policies via an SDN application. The application inputs the policies to the controller via APIs at the northbound level. The controller then transforms those policies into forwarding rules to be injected into the switches. Any update or modification is deployed in the same manner: the administrator interacts with the application instead of manually modifying policies in each node as in traditional networks. However, the SDN controller is a single point of failure, and redundancy should be enforced in an SDN network to avoid network shortage in case of a controller failure. IP routing protocols, on the other hand, provide reliability and fault tolerance.

Furthermore, SDN might not be the best choice for routing across AS's. In fact, network islands are owned by different authorities. In this case, a central controller (or central control plane) is not feasible, since different admins will have different set of policies. Also, they also may not want to share the network control with other entities for security reasons, for example. Therefore, traditional inter-domain routing such as BGP remains an appealing solution for inter-AS routing especially that it has been proven to scale over large networks and it offers granular control over policies.

Switching can also be improved with SDN, even though it occurs at the limited LAN. SDN permits greater control and security by manipulating the forwarding ports of the switches. SDN avoids broadcasting packets over the LAN, which is vulnerable to eavesdropping and mapping attacks. It also cancels the switch self-learning procedure, which is vulnerable to MAC spoofing attacks.

Given the benefits of each type of technology, a good approach would be to deploy both in the network to combine their advantages. However, this type

of hybrid network was spontaneously formed, not for the opportunistic reason stated previously, but as a natural consequence of shifting from a well-established technology (legacy IP) to a new one (SDN).

2.5 Overview of Centralization and Decentralization in Different Areas

However, centralization and decentralization of control exists beyond networks as we know them nowadays, it also exists beyond the area of networks. We rely on examples from the evolution of control planes in telecommunications, from political systems, and from natural intelligence to model the behavior of our hybrid system. Consequently, we studied many systems that exhibit the same interrelation between the two states of control. The most relevant systems are discussed below.

2.5.1 History and Evolution of Network Control Planes

Control planes in telecommunication evolved throughout the years based on the needs of each period. Learning the causes affecting these shifts allows us to model this behavior in our adaptively controlled hybrid design. In the 19th century, the communication system was extremely centralized. The telephone was introduced as a means to carry voice signals over long distances. Initially, it was conceived as a point to point system, where each caller and receiver were connected by a direct link. The bulkiness of the previous system gave way to the telephone exchange, which relied on a central switchboard, operated first by a human in charge of setting up the lines and connecting callers together, and later by ma-

chine switching equipment. In these circumstances, the communication system was extremely centralized. In addition to that, the network was only allowed to grow up to a certain size. With advancements in Public Switched Telephone Networks (PSTN), new techniques were needed to support a larger number of subscribers, because the centralized system was unable to respond to such requirements. Consequently, the control plane started to become more distributed using hierarchical routing across different levels, starting from local switches up to regional switches. The trend shows that control shifted from being extremely centralized when the network was very small to being more distributed to cater for scalability. With the invention of computers, another type of information, data, was to be transmitted. The first attempts were to transmit digital information of both voice and data over PSTN, using the Integrated Services Digital Network (ISDN), and later the Broadband Integrated Services Digital Network (B-ISDN), which uses the Asynchronous Transfer Mode (ATM) technology.

ATM Networks

Control planes started to become distributed with PSTN, and the trend continued with ATM, which was very popular in the early 90's. It was conceived and adopted as the infrastructure for B-ISDN, for carrying voice and data traffic, in an attempt to bridge the gap between computer networks and telecommunication. The control plane in ATM is distributed; the switches can autonomously initiate, terminate, or reject a connection. ATM provides the switches with all the information, needed to perform these operations, through the Private Network-Network Interface (PNNI) protocol. PNNI enables each ATM switch to have an overall view of the network. It populates the routing table, using Dijkstra's shortest-path-first algorithm [10].

IP Networks

Even though ATM was standardized for B-ISDN, its popularity started declining in favor of the IP protocol, which was offering better cost performance. The IP control plane was still distributed, but at a finer granularity; in ATM, decision functions were performed at the ingress switches via source routing; however, in IP, each router has the ability to route packets autonomously and along different paths. Routing algorithms populate the routing tables; they are categorized as Link State routing protocols or Distance-Vector routing protocols. Each router separately reproduces the complete topology of the network and computes the least-cost connectivity graph to all nodes in the network, employing Dijkstra's or any other shortest path algorithm. The operation of these protocols is comparable to that of the PNNI algorithm in ATM [10]. Routing protocols are also classified with respect to their zone of application: Intra-domain such as RIP and OSPF, and Inter-domain routing protocols such as BGP. In all IP routing protocols, the routers are the decision makers; they are in charge of signaling information to other nodes in the systems, as well as deciding when and where to transfer the packets, in an extremely decentralized control fashion.

Centralized Routing Control with SDN

With the substantial growth of the internet and cloud computing, network requirements changed. Traditional network infrastructures were less likely to deliver the needed performance, due to the distributed control across the network elements and the lack of standardization. Consequently, SDN emerged as an attractive new form of networks, which offers centralized control, ease of management and programmability. SDN deprives the switches and routers from the traditional decision-making function, and grants it to a central controller. The

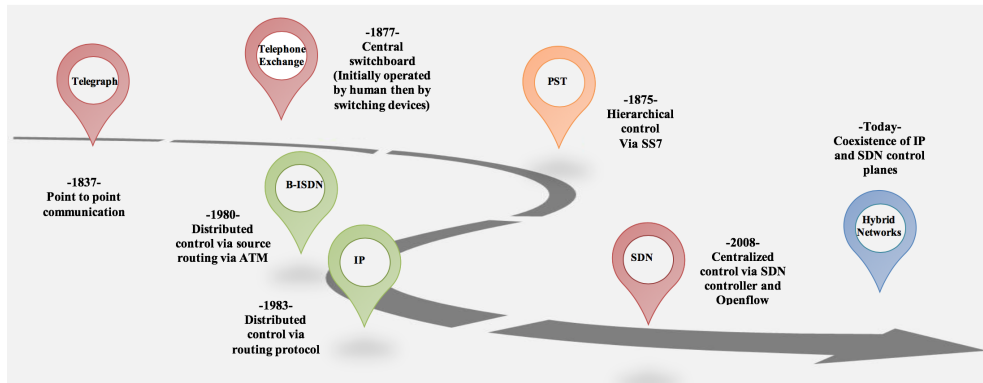


Figure 2.4: Evolution of the control planes; red: centralized control, orange: hierarchical routing, green: distributed control, blue: hybrid control

network elements are simply in charge of forwarding packets, following a set of flow rules injected into them by the controller. The controller builds a complete topology and link costs as it has a complete view of the network. Network applications and modules running on top of the controller process this data to make routing decisions and send them to the controller via the northbound interfaces. The controller injects the rules into the forwarding tables, called Flow Tables via OpenFlow (OF) [7].

Hybrid Networks

Hybrid networks deployment was a natural consequence of the availability of two types of control plane in networks, since full SDN deployment was infeasible for several reasons (financial, resource, expertise, etc.).

Many works emerged to foster the coexistence of both types. Some techniques relied on injecting fake link state advertisements by the SDN controller into a network composed of IP and SDN nodes, which allows combining the central control of SDN and the distributed route computation of link state protocols [18]. Other proposals considered translating the hybrid network into a logical SDN

by transforming the heterogeneous physical infrastructure composed of legacy and SDN nodes into homogeneous logical SDN nodes. This allows leveraging all benefits of SDN while avoiding the cost of deploying a full SDN network [19], [20]. In [21], the authors develop four NP-hard optimization problems, for choosing the optimal number of hybrid paths (paths that contain at least one SDN switch), and the number of needed SDN nodes in the hybrid paths. Additionally, coexistence was considered in [22], which proposed LegacyFlow as a solution to manage traditional IP equipment within an SDN environment. The approach consists of introducing a configuration layer that houses configuration modules specific to each equipment, in charge of translating the rules sent by to SDN. Other propositions deploy SDN on parts of the network while running conventional methods on other parts [23, 24, 25].

The previous proposals try to merge between centralized and distributed routing in networks that are composed of legacy and SDN nodes. However, it is worth mentioning that one of the most straightforward approaches is to employ hybrid switches. These switches support SDN and OF as well as the traditional network protocol stack. This type of switches is available in the market by many vendors [8]. This option was also spread in the literature as some researchers considered hybrid switches in their hybrid deployment. For example, the work in [26] tackled an OSPF/SDN network. However, their hybrid network was composed of OSPF-enabled traditional routers and hybrid switches that support both SDN and OSPF. The hybrid switches served as a gateway between routers and the controller: they were connected to the controller via the SDN/OF interface and communicate with the legacy switches via the OSPF interface. Additionally, Google started to experiment with hybrid deployments in their datacenters by deploying the B4 WAN, which is an infrastructure conceived to link Google's

datacenters from around the world [27]. A Routing Application Proxy which transforms routing entries into flow table entries was designed to integrate the SDN control plane with legacy control. Figure 1 shows the evolution of control plane as new technologies were invented.

Advantages and Analysis Each of the two types of control has its own advantages and disadvantages, and hybrid networks promise to emerge as the best of both. In fact, SDN removes the burden of distributed route calculation at the node level, which provides network administrators more control over their systems. However, Link state routing protocols are more scalable and respond better to changes in the routing path; they provide reliability and fault tolerance. Additionally, the SDN controller is a single point of failure, and redundancy should be enforced in an SDN network to avoid network shortage in case of a controller failure.

Consistency Issues In the light of the above, the hybrid network is of interest: it allows operators to exploit the benefits of SDN and legacy networks, and design an optimized form of a network that suits their requirements. However, hybrid networks include two types of control that might create signaling inconsistencies. The problem was initially identified in pure SDN networks [28], where inconsistencies might arise from distributed controllers. In [29], the authors tackle consistency of distributed instances of SDN controllers within an SDN network. This type of inconsistency occurs when the SDN control plane is implemented in a hierarchical fashion, or when it is horizontally distributed to account for large networks and to guarantee network scalability. The authors define a management layer that spans the different controlled domains, where each domain is made of a physical SDN controller. The authors show that performance is

degraded when the network applications are presented with wrong information about the network due to stale updates and inconsistencies between controllers' domains. Having different levels of control is challenging in pure SDN networks which run a single protocol, OpenFlow. The problem is surely aggravated in hybrid networks, which include different "types" of control protocols. In addition to the coexistence techniques developed in section II.D, other consistency methods were proposed to guarantee the well-functioning of hybrid control planes. In [30], Vissicchio et al. develop a framework for ensuring inconsistency-free network updates in Hybrid networks that are composed of nodes that support SDN and traditional protocols at the same time. In fact, the authors argue that the presence of two control planes in the network can cause routing inconsistencies and create problems such as loop and blackholes. The proposed algorithm ensures that these anomalies are avoided, making the incremental deployment of SDN in traditional networks possible. Additionally, the authors of [31] provide a general framework that solves the issue of control planes in hybrid routing. The work also supports consistency in an SDN network with different uncoordinated control hierarchy. It ensures coexistence of all control planes by modeling the theory behind it. Additionally, the authors propose a new taxonomy for routing control planes based on their input and output data structures. This classification was proven to be able to capture anomalies in coexisting control planes in such a way that any configuration that satisfies the conditions stated is guaranteed to be free of forwarding inconsistencies. The framework also gives guidelines for anomaly-free configurations and safe reconfiguration of a network with hybrid control planes.

2.5.2 Models of Centralized and Distributed Control

As previously discussed, hybrid networks were formed spontaneously as a result of the existence of two types of technologies with different types of control in networks. However, we can exploit hybrid networks and deliberately design them to integrate the advantages of traditional and SDN paradigms. In fact, this co-existence between centralized and distributed control exists beyond the area of communication networks. It is found in many human induced activities, such as administrative activities in banking or companies, or in politics and political systems. Additionally, the interaction between the two forms of control is also found as a natural phenomenon in nervous systems of living beings. Thus, these systems can provide guidelines on designing a similar control framework for networks.

Political Systems

Over time, political systems have shifted from being completely centralized, to completely distributed, up to a state where some power is centralized but some is not. At the same time, a stable distribution of power might be disturbed if current circumstances favored one type over the other.

Central versus Decentral Power The dynamics between centralization and decentralization are interesting to consider in the context of politics since they played a major role in history. The history of Europe best describes the issue as different political systems rose and failed. For example, the Roman Empire, led by the emperor, was extremely centralized. However, after its fall, Europe was left unprotected. This pushed landlords to protect their own environments, in a system where decentralization was at its peak. However, centralization found

its way back in the renaissance after the corruption of the church leaders was exposed. The status quo is believed to serve some balance between the two types of power [32]. However, the shift between states of distribution of power did not only happen sequentially over time. It is dynamically changing in current political systems as new circumstances emerge. In [33], the authors consider a framework for the centralization versus decentralization of power in countries such as the United States. Despite the fact that the modern world is moving more towards decentralization and division of responsibilities, it does not cancel the effect of the central power that some systems have (Federal governments), which can still control some aspects of the different sectors. The main guideline is to keep everything related to citizens as close to them as possible, which means that all services related to people are better handled by local administrations in a distributive manner rather than by farther state administrations. Also, in the context of the relationship between centralization and decentralization, many theories state that in order to realize efficient distribution of resources and attain economic stability, a robust central administration is to be fostered [34, 35]. On the other hand, some statements [36, 37] consider that decentralization should be fostered to limit to the malpractices of an incompetent central government. And finally, some views advocate the equilibrium between central and local governments where the appropriate balance should be carefully studied to reach the optimal equilibrium.

The United States (US) – An example In the US, the division of power between the different levels of governments is defined according to the nature of the activities. Basically, the federal government possesses power over critical national affairs, such as defense, national security and building an army or navy.

It also handles all relation with foreign countries such as the decision to enter a war or sign treaties, and national economic matters such as printing money. On the other hand, state and local governments conduct activities that are more related to their close environment. State governments are given the ability to decide upon laws for contract and family, and education, whereas local governments are concerned with all activities directly related to people, such as police and firefighting, health systems, public transportation, housing, and so on [38]. Although the division of power is clearly defined as such, the US witnessed shift of power back and forth, between central government and distributed state government, throughout major historical milestones such as WWI, WWII and the great depression. Prior to the 20th century, the federal government had limited power. However, centralization quickly gained power in times of crises in the early 20th century (WWI, WWII, and Great Depression) because there was no way to deal with the disorders other than to rely on the federal laws again. After the wars ended, decentralization started to be restored gradually [33].

Model Analysis The models extracted from the above serve as guidelines for a possible model in networks. For example, the emergence of the highly distributed society after the fall of the Roman Empire –which was the central and only power – insinuates the need to switch to distributed routing signaling in a network if the SDN controller fails. Also, allocating the decision to go to war or responding to crisis to the federal centralized government in the US can be mapped to switching the network to a fully centralized state in the event of network failure or attack at the links and nodes level. Moreover, the division of power based on the activity in the US can be mapped to a segregation of services in the network, where each service will be controlled centrally or not according to its nature. For example,

MPLS and tunneling protocols favor centralized control because it facilitates their implementation and operation.

Natural Intelligence: The Nervous System

The nervous system of living beings provides another good model for studying the interactions between states of control. In neuroscience, two types of nervous systems exist: diffuse and central. An organism with a diffuse nervous system doesn't have a brain or any central organ that controls it; it has uncoordinated distributed neurons in a mesh fashion that react even to minor stimuli. This is characteristic of lower invertebrates. For higher invertebrates, a collection of nervous cells is aggregated in a longitudinal form and a brain along with coordination interneurons. In vertebrates, the centralization of the nervous system is more pronounced as the nervous system center is clearly contained in the brain, the spinal cord. This center is in charge of coordinating all activities of the organism. The vertebrate nervous system has a centralized response to stimuli; it is composed of two subsystems: the central nervous system (CNS) and the peripheral nervous system (PNS). The central nervous system is composed of the brain and the spinal cord, which are the decision makers. The peripheral nervous system is composed of all sensors that capture events and motors that induce a certain action. The PNS is in charge of relaying all information captured to the CNS, and executing the commands it receives from the CNS in response to the information sent. Although the vertebrate nervous system is classified as centralized, some of the responses are not emanating from the central organ: the brain. Sensory nerves relay a stimulus to the spinal cord which is considered as an extension to the brain. Its function differs from that of the brain because it is responsible for a specific type of behavior, notably the reflexes. This allows the

body to respond quicker to critical events by taking an action before the signal reaches the brain. Reflexes are classified under two types: reflexes involving 3 or more neurons or long reflexes, such as when a nerve is hit in the knee the leg automatically reacts, or less than two neurons, for example the automatic blink of the eye if the cornea is disturbed. An interesting observation is that we are not aware of any event which signal does not reach the brain, as in the case of reflexes. Eventually, the brain will receive and analyze the signal but the reflex would have been already carried out. This characteristic allows reflex actions to occur relatively quickly by activating spinal motor neurons without the delay of routing signals through the brain [39, 40, 40].

Model Analysis The nervous system model also shows some guidelines although they differ from the rules followed in political science: the nervous system has a different reaction to danger or crisis. In fact, in the administrative model, the strategy to respond to crises is to go fully centralized and give full control to the federal government. However, the nervous system responds to dangers via reflexes which are the non-centralized decision-making aspect. The difference is basically related to the speed of action to be taken versus its skillfulness. In the first scenario, the political response to political or financial crisis is not time critical but rather necessitates strong coordination and shrewd decisions. Whereas when the body faces a danger, for example touching a hot surface, a quick reaction is needed to avoid damage. In this case, the signal is automatically decoded in the spinal cord and the action of moving the finger off the surface is directly undertaken. Analysis of the signal and action is eventually accomplished by the brain after the reflex is done. This guideline can be used in our design, because it proposes categorizing the potential dangers on the network between

time or robustness critical, and consequently decide on employing distributed or centralized control, respectively.

Other Applications

Security: Blockchain Technology The question of central versus distributed also infiltrated the security domain. The Blockchain technology was introduced as an alternative to the existing centralized regulatory body, which controlled all transactions carried out over the Internet. It was also conceived as a replacement to the Public Key Infrastructure (PKI) system for the advantages it provides in terms of encrypting, storing, and managing public keys. In fact, the distributed characteristic of Blockchain makes it fault-tolerant, and single point-of-failure averse. The threat of trusted third party is mitigated with Blockchain, which alternatively relies on cryptographic hash functions to verify and validate transactions. Additionally, it provides more automation because human intervention is no longer needed. It also promises better manageability and more cost-effectiveness [41, 42]. Given the advantages of Blockchains, research was channeled through implementing the previously centralized PKI system using the new decentralized technology. Certcoin [43] was proposed as a blockchain-based PKI, where the identities and public keys are coupled with the corresponding instruction, for example register, revoke, update and so on, and stored in the blockchain ledger. They are consequently processed via the normal mining or verification of the blockchain network.

5G: D2D communication We can also see the interaction between centralized and distributed aspects in 5G with the recent introduction of device-to-device communication (D2D). In fact, with D2D, devices are allowed to communicate di-

rectly with each other without relying on the backbone network if they are in each other's proximity. D2D communication was rarely studied in cellular networks until the emergence of 5G where it gained more weight. In fact, 5G introduced new requirements such as reduced latency, context aware services along with an exponential increase in number of applications and connected devices. This emphasized the need for D2D because it can reduce the burden on the base station (BS) due to the increased traffic, which reduces delays and delivers better performance. Furthermore, users can rely on D2D to exchange critical information in case of emergency when the BS infrastructure and communication network are down due to disasters. This is similar to reverting to distributed routing when there's a given problem on the network, for example, if the controller goes down [44, 45].

In light of the above, we defined rules that trigger a system to move from centralized to distributed control. The control tasks can be allocated among distributed and central control planes as shown in Figure 2.5.

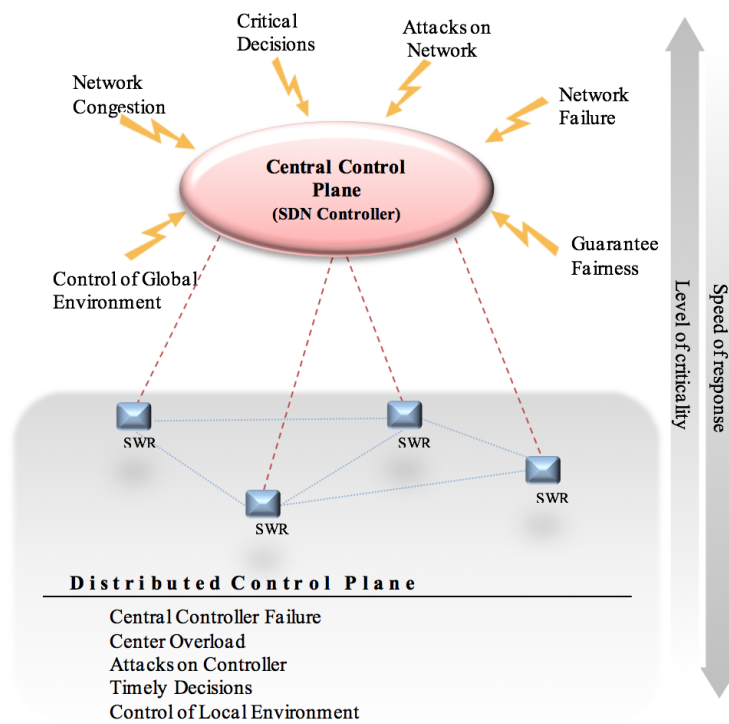


Figure 2.5: Allocation of control tasks

Chapter 3

Effect of Centralized and Distributed Control on Networks

3.1 Hybrid Control Planes in Networks

Although SDN is emerging as a new networking paradigm to overcome the shortcomings of traditional 7-layered networks, the shift from traditional networks cannot happen overnight. Consequently, three forms of networks are available today: traditional networks built using Layer-2 switches and Layer-3 routers implementing distributed control protocols, pure SDN with centralized control and programmable control and data planes, and hybrid traditional-SDN networks, which are combinations of the previous two and are typically the result of partial SDN deployment. In this section, we refer to the last type of networks as Hybrid Networks (HN). HN are already being implemented, and many techniques are developed to support their deployment. For example, the work in [46] introduces HN and categorizes them with respect to their behavior. The first category of HN is composed of two types of equipment, SDN and traditional nodes that are usu-

ally grouped by routing islands. The second category segregates network services and deals with each type of services with either SDN or traditional paradigms. The nodes in these networks can be of either type, or a node that supports both type of paradigms. In the third category, different classes of traffic are processed with either SDN or traditional protocols. Finally, the last category integrates both paradigms in one framework, where SDN is in charge of all the services, sits on top of the traditional protocols, and controls the distributed traditional protocols. The HN types are also compared in terms of cost, scalability, robustness, etc.

3.1.1 Integrating Traditional Routing Protocols with SDN

Hybrid deployment is gaining interest for many reasons; first of all, full SDN deployment may be infeasible for several reasons (financial, resource, expertise, etc.), but more importantly, stakeholders realize that a better routing scheme would be generated by integrating the advantages of traditional and SDN paradigms instead of completely replacing the first. In fact, traditional routing has been developed for many years, proven globally for years in production environments, and a lot of research has been invested in the field, all of which can be exploited along with the offerings of SDN to improve routing. Many works emerged to that end, each providing a different combination of the features.

In [18], Vissicchio et al. presented Fibbing, a routing technique that joins the central control of SDN and the distributed route computation of link state protocols. The authors argue that link state routing protocols are more scalable and actually respond better to the changes in the routing path, and thus route computations should be performed by these protocols. On the other hand, they take advantage of the centralization provided by SDN to control the distributed

protocols. The Fibbing (SDN) controller is fully informed about the network topology, and it is capable of computing an Augmented Topology (AT) of a network, which corresponds to the real network topology extended with fake nodes and altered link weights. The authors proved the existence of such an AT by translating the network into Directed Acyclic Graphs (DAGs), casting them into sets of constraints and then computing a new AT to address this set of requirements. The Fibbing controller feeds the AT to any link-state routing protocol by fabricating fake link state advertisement (LSA) to advertise the fake routes. This indirectly allows the controller to control traditional routers and redirect traffic in the desired direction. The paper presents three scenarios where Fibbing can be employed: traffic steering, failure recovery after link breakage, and load balancing. The operation of the approach is simple; when the controller wants to enforce a route, it advertises a fake node on that route having the destination as a next hop with minimal cost. Upon receipt of the advertisement, routers save the fake route as a valid best path to destination. The link state algorithms will force traffic through the fake route which physically corresponds to a real path with different weights. This technique can be extended to cover cases where the network is formed of traditional routers and SDN switches since it allows forcing flows to go through the right hybrid path.

A similar “lies-injection” approach was employed for switching in [47]. Jin et al.’s method did not only present an approach for switching in hybrid networks, it also showed that a hybrid network can offer the full advantages of SDN if the SDN to traditional switches ratio and the location of the switches are optimal. Just like Fibbing, the Telekinesis controller injects fake MAC addresses via the PacketOut function of OpenFlow. A switch having learned the new MAC address inserts in its forwarding table a forged entry that associates a port to a spoofed

MAC. Telekinesis becomes able to control all the SDN and non-SDN switches interfaces, and to steer traffic through them.

HybridFlow [20] is another attempt to reconcile between legacy control plane and the SDN plane, and enable the hybrid network to operate. HybridFlow transforms the hybrid network into a virtual SDN, where legacy switches are clustered with one SDN switch to form a virtual SDN switch. All physical ports are translated to virtual ports, which are exposed to the control applications. Since these applications can only see virtual ports of virtual SDN switches, the controller is in charge of translating the control rules to match the real hybrid network.

Partial SDN deployments are also considered in [19], where the authors propose “Panopticon”, a framework that facilitates the operation of HN. Panopticon abstracts the hybrid network into a logical SDN network, by transforming the heterogeneous physical infrastructure composed of legacy and SDN nodes into homogeneous logical SDN nodes. Consequently, the network is modeled as a collection of paths from source to destination, and offers all advantages and programmability of SDN as long at least one SDN switch exists on each path; the more SDN switches on a path, the more it is controllable. The paths consist of VLAN tunnels that segregate the end points from the physical network. The tunnels are forced to pass through one or more SDN switches, whose location and number are chosen based on a cost optimization problem. This framework allows operators to leverage all benefits of SDN while avoiding the cost of deploying a full SDN network.

In [21], the authors develop four NP-hard optimization problems, for choosing the optimal number of hybrid path (path that contain at least one SDN switch), and the number of needed SDN nodes in the hybrid paths, while keeping the

cost below a certain target, and then minimizing the cost with respect to each of the number of hybrid paths and the number of SDN nodes in the paths. This scheme, allows leveraging all the benefits of SDN while avoiding the incurred cost of totally replacing traditional network with pure SDN. Additionally, coexistence of both types of appliances was considered in [22], which proposed LegacyFlow as a solution to manage traditional IP equipment within an SDN environment. The approach consists of introducing a configuration layer that houses configuration modules specific to each equipment. These modules are connected to legacy data path block in charge of translating the rules sent to the SDN controller (to which it is connected to) to vendor specific rules corresponding to the underlying type of switch via the corresponding configuration module.

3.1.2 Routing Islands

The above approaches tightly mix SDN with traditional practices; both paradigms coexist within the network. Other propositions deploy SDN on parts of the network while running conventional methods on other parts.

In [23], Hall et al. argue that SDN will replace the intra-domain routing protocols such as RIP and OSPF, but not BGP. Routing inside an island is handed-off to the SDN central control plane, which implements internal routing via a routing application, while routing across islands is still performed via BGP. The paper focused on the BGP/SDN interface and introduced the Quagga for SDN Module (QuaSM) to support this integration. QuaSM is an SDN extension to the Quagga Routing Suite. It actually manages connections between BGP routers and handles all decisions related to route selection and route advertisement. Therefore, EBGP is still deployed across AS's but supervised from a central SDN entity. On the other hand, IBGP can be fully replaced by QuaSM since this latter is ex-

posed to both BGP and SDN interfaces. It learns routes from EBGP and informs internal border routers via SDN. The centralized control plane is thus aware of internal and external information. This technique can be used to integrate SDN islands with traditional islands in the network. It can also be used within large pure SDN networks comprised of many AS's to connect them together.

Kotronis et al. presented a similar idea in [48]. They exploited the concept of centralization of control in SDN, and proposed to outsource routing within AS's to an external trusted entity, the contractor. The difference between this approach and SDN lies in the fact that the SDN controller is part of the network, managed by the same administrator and spans one physical network but many virtual networks, whereas the contractor is an independent entity that could serve many physical networks. The contractor is in charge of managing all routing within an AS based on Service Level Agreements (SLAs) agreed upon with the AS administrator. SLAs allow it to serve many AS's at once. It also handles routing across AS's by managing EBGP sessions; it acts as a man-in-the-middle, as all BGP messages pass through it allowing it to make global routing decisions. The contractor hence introduces the possibility of eliminating BGP for two reasons: first of all, when the contractor manages internal routing for many AS's and shares SLAs with each, it can also manage routing across those AS's and cross-match the pair of SLAs to figure out the rules to be enforced. And second, the contractor is free to use any method, BGP or any other new protocol that complies.

In [24], a contrasting approach was considered, where SDN was employed to manage AS's border to border connection, while OSPF was used as an intra-domain routing protocol. In their work, the authors use SDN control plane to manage distributed protocols within the AS's. The method relies on subdivid-

ing of the network into OSPF islands that are interconnected by SDN border switches. The centralized control plane is aware of the whole network, because all OSPF advertisements are relayed to the SDN controller by the border SDN nodes. Consequently, this latter can alter the advertisements and push them back in the network via the border switches, which will eventually be flooded in the neighboring OSPF island. The authors also develop an optimization algorithm based on ILP that provides the optimal partitioning of the network with balanced subdomain size. Results show that this form of partitioning outperforms networks with only IP routing, while providing similar performance of pure SDN.

3.1.3 Hybrid Switches

The previous proposals try to merge between centralized and distributed routing in networks that are composed of legacy and SDN nodes. However, it is worth mentioning that one of the most straightforward approaches for Hybrid control planes is to employ hybrid switches. These switches support SDN and Openflow as well the traditional network protocol stack. This type of switches is available in the market by many vendors [8].

This option was also spread in the literature as some researchers considered hybrid switches in their hybrid deployment. For example, the work in [26] tackled an OSPF/SDN network. However, their hybrid network was composed of OSPF enabled traditional routers and hybrid switches that support both SDN and OSPF. The hybrid switches were connected to the controller via the SDN/OF interface and communicate with the legacy switches via the OSPF interface.

Additionally, Google started to experiment with hybrid deployments in their data centers [27], by deploying the B4 WAN, which is an infrastructure conceived to link Google's data centers from around the world. Their architecture

is composed of three layers: the highest layer spans network applications such as traffic engineering. The middle layer is composed of site controllers, which are connected at the lower layer to switches that run legacy routing protocols augmented with an Openflow agent. Each hybrid switch is connected to a cluster of BGP router that connects the B4 infrastructure to the sites. The centralized control of SDN was exploited for traffic engineering where almost 100% bandwidth utilization was achieved. The work included a Routing Application Proxy that was designed to integrate SDN control plane with legacy control. The application basically transforms routing entries into flow table entries using the ECMP hashing technique.

As hybrid networks deployment became more popular, a test bed was needed in order to validate the different ideas through experiments. SDN is already widely emulated and simulated via Mininet, but no environment for hybrid testing is readily available publicly. OSHI was presented as an open source experimental tool for emulating hybrid networks and hybrid nodes [49]. OSHI builds on Mininet and adds nodes that support OVS and IP routing. Its operation is inspired from the IP/MPLS integration and the introduction of tags or labels in order to differentiate the types of traffic. In fact, the OSHI node segregates IP traffic from SDN traffic by appending VLAN tags to flows. The method is feasible when the underlying links do not implement VLAN or when OSHI uses a set of reserved tags – to avoid VLAN tag collisions. The OSHI node implements a regular OVS, augmented by an IP routing/forwarding module to which it is connected by virtual ports. Upon receipt of a packet, OVS examines the tag, and forwards it along the normal SDN path if it is SDN traffic or to a virtual port if it is IP traffic. Consequently, that packet is processed by an IP routing daemon, which is Quagga in this tool. OSHI also offers a graphical user interface for topology

management and a set of performance Measurement Tools (OSHI-MT).

3.1.4 Consistency of Control Planes

In the light of the above, the SDN network is attractive: it allows operators to exploit the benefits of SDN and legacy networks, and design an optimized form of a network that suits their requirements. However, hybrid networks include two types of control that might create signaling inconsistencies. Therefore, many works that address this issue were conducted. In Google's B4, an application that translate information between the two was developed, whereas in OSHI the two control planes were being updated simultaneously.

Initially, the problem was identified in pure SDN networks [28], where inconsistencies might arise from distributed controllers, even before hybrid networks were considered. In [29], the authors tackle consistency of distributed instances of SDN controllers within an SDN network. This type of inconsistency occurs when the SDN control plane is implemented in a hierarchical fashion, or when it is horizontally distributed to account for large networks and guarantee network scalability. The authors define a management layer that spans the different controlled domains, where each domain is made of a physical SDN controller. The authors show that performance is degraded when the network applications are presented with wrong information about the network due to stale updates and inconsistencies between controllers' domains.

Different levels of control are degrading in pure SDN networks which run a single forwarding protocol. The problem is surely aggravated in hybrid networks, which include different types of protocols, having either distributed or centralized control. In [30], Vissicchio et al. develop a framework for ensuring consistency-free network updates in Hybrid networks that are composed of nodes that support

SDN and traditional protocols at the same time. In fact, the authors argue that the presence of two control planes in the network can cause routing inconsistencies and create problems such as loop and blackholes. The proposed algorithm ensures that these anomalies are avoided, therefore making the incremental deployment of SDN in traditional network possible. Additionally in [31], the authors provide a general framework that solves the issue of hybrid routing control planes- centralized versus distributed. In fact, with the emergence of SDN hybrid networks that support both Openflow and legacy routers, a framework was to be provided to ensure consistency among the control planes of the different protocols. The work also supports consistency in an SDN network with different uncoordinated control hierarchy. It ensures coexistence of all control planes by modeling the theory behind it. The authors also present a general model for a hybrid router that can function with any control plane. Additionally, they propose a new taxonomy for routing control planes based on their input and output data structures. This classification was proved to be able to capture anomalies in coexisting control planes in such a way that any configuration that satisfies the conditions stated is guaranteed to be free of forwarding inconsistencies. The framework also gives guidelines for anomaly-free configuration and safe reconfiguration of a network with hybrid control planes.

3.2 Effect of Different Types of Control on Network Services

Hybrid networks gained a lot of interest first for augmenting advantages of IP routing with advantages of SDN, and second for constituting a solution for incremental deployment of SDN. However, the Internet infrastructure implements

many services other than routing. Network services differ in their requirements: some services perform better with a centralized signaling plane, and some do not. For example, compared to legacy networks, SDN was proven to provide a better infrastructure for MPLS. Whereas the distributed control plane favors middle-boxes deployment. Consequently, hybrid networks (HN) became an effective solution that exposes two types of control planes in the network, so that the different network services can be implemented with the control plane that is the most favorable for their operation. Accordingly, we showed below how different services are implemented in the three types of networks, pointing out the advantages and disadvantage of implementing each service with the different technologies.

3.2.1 Middle-boxes

A middle-box is a network element that carries out a particular activity, such as a firewall or a proxy. The following subsections cover some of the most common ones.

Firewalls

Traditional Networks A firewall is a popular type of middle-boxes, responsible for enforcing access policies. In today's network, a firewall can also be implemented on the application layer running on hosts. However, these will not be considered in this paper as only network layer Firewalls are of interest. A network firewall grants or prevents access from an outside "world" to a given network based on a set of predefined rules that states if the outsider is to be trusted or not. These rules are generally set by the administrator and they are used to filter packets based on the port and IP addresses of source or destina-

tion, TTL or protocols. Firewalls are usually classified as stateful or stateless. Stateful firewalls monitor and track the states of the TCP or UDP connections that pass through in the aim of speeding up the processing, whereas stateless firewalls don't. A router can have firewalling capabilities if it is in charge of passing packets across different networks [[50], [51]].

SDN An SDN switch inherently acts like a firewall since it has the capability to forward or drop flows based on the rules that the controller injects into its forwarding table [52]. The rules are defined and computed by a firewall application running on top of the controller, which communicates with that latter via REST-like northbound APIs. Many controllers provide built-in Northbound API for Firewalls such as OpenDaylight [53], Floodlight [54], and Ryu [55]. The firewall can also be implemented in the controller; POX offers the possibility to write a Firewall module in it, as in [55] or [56]. The controller then enforces the rules on the switches via OF. Researchers went further towards building complete firewall frameworks that offer full access-control in SDN. For example, FLOWGUARD [57], built in floodlight, provides two mechanisms to increase the reliability of firewalls in SDN: increase violation detection by examining the flow at the switch level, as well as tracing the flow path and better violation resolution.

HN Openflow together with the controller rules and the SDN switches can create sufficient SDN firewalls. However, this will not provide full control over the network, which is exactly why we saw the emergence of enhancements to that scheme as in [57]. In fact, well-known SDN switches such as Cisco's Nexus 9000 or the OVS switches function as stateless firewalls [52] and thus, they cannot completely cover all functionalities of traditional firewalls, more specifically the stateful firewalls. Therefore, much work in the literature addresses the issue of

firewalls in SDN suggesting that one is better off integrating traditional firewalls and middle-boxes in SDN networks rather than completely discarding them [58].

Analysis Even though SDN provides flow filtering, it does not mimic the operation of a traditional firewall, as discussed above. However, SDN allows the implementation of distributed firewalling. Firewall Switches can be dispersed within the network and easily managed by the controller: they can be configured separately or configured as multiple instances of a single firewall. In fact, a topology module running on the controller creates a top view of the network with the exact location of the FS. The controller can then execute a firewalling module, which synthesizes rules based on the locations of the firewalls and the administration policy plan. The controller inserts the rules in the flow tables of the FS. Furthermore, since the FS are Linux based, they can host any piece of software such as a firewall module. The module can be connected to the virtual port of the OVS. This module can be programmed to enable them to perform state-full decisions, which will turn the switch into a legitimate firewall.

Requirements of the given network are crucial in order to decide whether to go for SDN or for a hybrid setting. The type of network, its size and its usage dictate these requirements. A small private network would not need more than a filtering functionality provided by SDN, whereas a university campus would definitely necessitate stronger firewall functionalities provided in a distributed manner across the network.

Proxy

Traditional Networks A proxy server has many functions, ranging from monitoring and control, performance enhancement to security tasks. A proxy is

thought of being a server that caches resources in order to improve the response and more generally the performance over the Internet. However, it can also implement a content-control platform along with authentication, which allows controlling the information and resources on the network and logging the corresponding user. This also means that the proxy can actively listen and record data and activities occurring in the network [59].

SDN The implementation of a proxy server with SDN is similar to that of a firewall. A proxy application is in charge of creating rules in response to the different scenarios. The controller acquires these rules via NB interfaces and injects them into switches, or proxy-switches. Similar work was proposed in [60] by Anderson et al. by relying on the SDN architecture to implement an efficient and scalable proxy for cloud providers. The proxy is deployed on an actual SDN switch connected to the manager, which actually resides on the controller. The manager controls the resources and decides on their allocation based to demand. These decisions are translated into OpenFlow actions injected into the proxy-switch. As all traffic is directed to the proxy, each flow is compared to a rule that dictates its fate. Given the nature of the rules installed on the proxy-switch, it can implement traffic redirection, load balancing and many other functions. Similar work was proposed in [61], whereby an ARP proxy module was implemented on an SDN controller to improve performance.

HN The scenario presented in the previous section and illustrated in [60] would also operate in hybrid networks. In fact, given the proxy-switch virtual IP address, a network operator can force all nodes in a network to send their packets to that IP regardless of the final destination. As the proxy-switch receives packets, it will match them against the flow rules inserted in its table. Another scenario for

proxies in hybrid networks happens when the proxy is itself a traditional server deployed in an SDN environment.

Analysis A proxy server is as easily implemented in hybrid networks as in traditional or SDN networks. It is a service that does not have favorable architecture since it constitutes an access point with a static location with respect to the services it controls. It is easily deployed on extremities of a network to grant entrance to the Internet island or other servers.

Deep Packet Inspection (DPI)

Traditional Networks Deep Packet Inspection (DPI) differs from the normal network filtering since it inspects the payload as well as the header. The increased complexity of DPI with respect to header-based filters enables them to detect, recognize and categorize packets that otherwise would go undetected. DPI detects viruses, non-conformity to protocols or intrusions based on which it makes a decision on how to process the packet. This technique is usually employed by large companies, ISPs and governments [62].

SDN SDN deals with network traffic based on flows and not individual packets. However, DPI is still possible since the controller can inject the “packet-in” rule in a switch in such a way that received packets are sent to the controller [63]. A DPI application or controller module will then examine the packet and return it to the switch. Based on this concept, Li and Fu proposed a DPI extension to OF in an SDN environment [64]. They however implemented the DPI system as a module on the controller itself. The DPI module receives the packet, processes it, compares it to the policy table, sends the packet back to the switch and updates the switch’s flow table. The authors claim that implementing DPI as

a controller module gives a more robust solution than implementing DPI as a network application since it decreases the communication overhead between the controller and the application.

HN DPI can be implemented in a hybrid network by using legacy middle-boxes. Legacy and SDN nodes can coexist based on the routing and switching techniques for hybrid that were previously discussed [18]. Nonetheless, the DPI system can also be implemented as an SDN application on top of the controller. The DPI will thus be enforced only on some SDN switches which forward packets to the controller for processing.

Analysis Even though DPI can be implemented in SDN, the centralized flow based architecture might not be the best option for this kind of service. In fact, DPI requires an in-depth inspection of the suspicious packets, which is not supported by the switches. Consequently, packets should be sent to the controller to be inspected, leading to the overload of the controller. Another SDN practice is to perform DPI on the first packet of the flow. In this scenario, only the first packet of a flow is sent to the controller, which injects the rule to apply to subsequent packets of the flow based on the DPI performed by a module or an application. This scheme is however vulnerable to attacks since a node can fabricate the first packet of a flow to be harmless and then tamper with the subsequent packets. DPI can also be implemented on a switch by adding a Linux DPI program. The controller then installs rules to redirect the packets that match a certain profile to this DPI switch. This solution solves overloading the controller, but does not differ from deploying a traditional DPI middlebox in a network. The DPI middlebox can be connected to an SDN switch that normally forwards flows along the SDN datapath but relays packets to the DPI interface

port according to some rule. Therefore, we can argue that SDN does not simplify the implementation of DPI, and that DPI in traditional networks is as efficient as in SDN.

Intrusion Detection and Prevention Systems

Traditional Networks An Intrusion Detection System (IDS) is also a device deployed in a network to monitor all traffic. It is different from a firewall since its aim is to detect abnormal behavior rather than just inspecting incoming traffic as the firewall does. In addition to preventing attacks from the outside network, IDS looks for and mitigates intrusions from within the network itself, and signals an alarm in response to any suspicious activity. An IDS system either operates online in real-time, or offline on collected data. It is composed of three main entities: a manager, a list of malicious signatures, and monitoring sensors. The sensors compare any dubious activity to the list of signatures and send an alarm to the manager in case of a match [65]. An intrusion prevention system (IPS) is similar to an IDS because it also monitors traffic. However, the main difference between the two is that IPS is able to take action after detecting the security breach. For example, in addition to signaling an alarm, IPS can also drop the packet, block flows or reset TCP connections in response to intrusions.

SDN Literature shows an emerging interest with IDS in SDN as many ideas emerged in that area. However, many of these are concepts that are yet to be fully developed, which indicates that this area is still in its initial stage. Many proposed IDS schemes did not target SDN for the sake of finding IDS implementation alternatives for SDN, but given the different management and monitoring benefits that the latter delivers. Lobato et al. [66] provided an example of such

architectures by implementing an IDS system, FITS. The controller is in charge of redirecting the packets to the IDS unit by injecting the corresponding rules in switches. The IDS processes the packet and raises an alarm to the controller whenever a malicious flow is detected. Then, the controller appends a drop action to the forwarding rule associated with that flow. TIPS [67] is another example of IDS that is not initially conceived for SDN, rather it takes advantage of the centralized nature of SDN to implement a robust and efficient IDS. The IDS scanners lie between two SDN switches which constitute a separation layer between the outside network and the internal network. All traffic is redirected to the switches which send packets to the scanners. This scheme provides load balancing among scanners and defends from attackers aiming to tamper with them.

HN In [66] and [67], similar schemes were extended to hybrid networks. In fact, as long as the IDS is connected to an SDN controller and a hybrid SDN switch, the type of devices that exist on the network becomes insignificant. Traditional routers can communicate with the hybrid SDN switch as it presents two interfaces, a traditional one and an OpenFlow one. The IDS scheme should be implemented in SDN and enforced via the SDN switch only.

Analysis Intrusion detection System deployment in SDN is similar to that of DPI. Malicious packets are either sent to the controller that has an IDS module, or to a traditional IDS scanner. Similarly to DPI, the IDS SDN version induces overhead to the controller if the first option is used. Also, implementing IDS is as efficient as adding a legacy scanner to the SDN network. Therefore, SDN does not provide additional advantages to traditional IDS. However, SDN is a better architecture for IPS. In fact, the ubiquitous SDN controller provides a better platform for intrusion prevention. IPS is limited in traditional networks

as these networks do not behave well in real-time. Any change applied to the network needs to propagate along all nodes, and this effect drastically grows with the size of the network. SDN, on the other hand, delivers a real-time response; the controller has a full view of the network, topology maps are generated via powerful network applications and rules are immediately updated in the switches. When a security alarm is generated, the IPS module collects information from the topology module and the switch that raised the breach, computes new rules and changes routes if necessary. The controller can thus react to intrusions instantly.

Middle-boxes in Hybrid networks

A special interest was given to the integration of legacy middle-boxes with SDN for many reasons. First of all, middle-boxes are expensive, and companies are not ready to dispose of functioning middle-boxes and invest in new SDN applications. Second, big stakeholders fear grouping all critical and security functions in one central component. The controller might become overloaded with all these functions which would constitute a threat to the quality of service of the network. In [58], Fayazbakhsh et al. proposed a method that incorporates middle-box management to the SDN management plane, instead of completely changing their architecture or eliminating them. “FlowTags” are added by middle-boxes to the headers of flows that pass through them, and constitute the interface between middle-boxes and SDN controllers and switches. These tags are then decoded by SDN switches and used along with the original OF forwarding entries. An important aspect of “FlowTags” is that minor modifications are applied to middle-boxes and to SDN entities; those latter still use OF for their operation. The system was tested for performance with Mininet, and built as a POX module on top of the controller. Results showed that the overhead induced by the addition of the tags

is negligible (less than 1%), and that its operation is linear with respect to the network size. SIMPLE [68], or Software-defined Middle-box Policy Enforcement, takes advantage of the characteristics of an SDN environment and applies it for middle-box management. It allows the integration of existing middle-boxes to SDN. SIMPLE provides network operators an interface to specify policies at a very high level; these policies are then translated into forwarding rules applied on the underlying infrastructure. SIMPLE requires three inputs: the policies, the network topology and the resource constraints, such as memory, CPU, size of routing tables, etc.; it also requires few additions to the data plane, such as tags and tunnels, in order to account for the overhead of the new method. SIMPLE runs offline ILP to create forwarding rules given the size of TCAM tables. This algorithm runs offline since network topologies are unlikely to change on a small timescale. Load balancing on the other hand, is accounted for through online LP, since the flow rate is frequently changing overtime. SIMPLE proved to be effective when implemented with POX on Mininet and Emulab. Nimble [69] is another module for the integration of legacy middle-boxes in SDN networks. It is implemented in POX, and is composed of 3 blocks: a resource manager, a dynamic handler and a rule generator. An administrator inputs the processing logic into NIMBLE. These rules along with location of the middle-boxes allow NIMBLE to define the routing path for traffic through middle-boxes when necessary. In this design, each middle-box is connected to an SDN-enabled switch that has both SDN and traditional interfaces. It is worth noting that service chaining plays an important role in all the above methods. Tags appended to the flows identify the sequence of the middleboxes a packet should be routed to. As packets flow through the chains, tags are popped off to avoid loops.

3.2.2 MPLS

Traditional networks

MPLS or Multiprotocol Label Switching was first conceived as a solution to the complexity of ATM. It was also designed to mitigate the routing shortcomings of IP, such as the slower routing caused by independent decisions at each router and long IP headers, its connectionless state which prevents any QoS service, and so on. MPLS is a data transport technique that sits between layer 2 and 3, forming a middle point between packet forwarding (IP) and circuit-switching (ATM) protocols. The protocol is based on assigning labels to IP addresses, which creates a virtual end-to-end tunnel over any transport protocol. An MPLS network is composed of the following entities: Label edge routers (LER), in charge of appending MPLS labels to packets entering the network and popping labels off the packets leaving. They reside on the edge of the network. These routers should be able to process all types of networks (ATM, Ethernet, etc.) on their ports. Inside the network, label switching routers (LSR) are in charge of transporting the packets with high speed through a Label Switching Path (LSP), using the labels appended to each packet [70].

MPLS Labels

MPLS labels are one of the most important factors in MPLS since they encompass all needed information for forwarding the packet. In other words, only the labels are examined to construct the forwarding decision. These labels are composed of 4 fields as shown in figure 3.1.

Many MPLS labels could be appended to a datagram, forming a stack. In

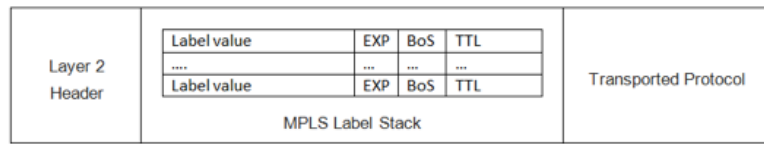


Figure 3.1: MPLS label

each MPLS label, the actual label is the first 20-bit field, followed by 3 experimental bits that are used for QoS. The next bit (bit 23) is the Bottom of Stack bit. This latter is set to 1 if this label is the last in the label stack. The last group of bits represents the time-to-live, having the same function as the TTL in a usual IP header. These labels are provided to LSRs and LERs via the application layer protocol called Label Distribution Protocol (LDP). Figure 3 also shows the positioning of the MPLS label between the headers of layers 2 and 3 [70].

MPLS Operation

In MPLS, labels are associated to forwarding equivalence class (FEC), which is a packet flow, or a group of packets flowing from source to destination following one path. In LDP, the mapping between FEC and labels are initiated by downstream routers, and are used to create the routers' label information base (LIB) table, along with the corresponding information about input and output ports. When a packet reaches an MPLS domain, the LER determines the corresponding FEC, looks up in its table the appropriate label (or label stack), and appends it to the packet. The router also uses the next hop for this FEC to forward the packet in the MPLS network. When LSR receives the labeled packet, it inspects the MPLS label (or the top label of the label stack), looks up the label against its table, and swaps it with the corresponding outgoing label. The packet is then

forwarded along the output port. When the packet reaches the LER on its way outside the MPLS network, the packet would have one MPLS label left. The LER pops off the label leaving the packet with a normal IP packet.

SDN

MPLS was integrated with SDN for the first time in 2011, when MPLS support was added to OpenFlow version 1.1 [71]. MPLS functions were upgraded throughout the subsequent versions of OF, and new actions such as Pushing MPLS Header, Popping MPLS Header, Setting MPLS TTL and Decrementing MPLS TTL, were added to the action list of OF. MPLS operation became straightforward: when a packet reaches an ingress switch, the controller orders the switch to append an MPLS tag to the flow. As the flow is circulating in the MPLS domain, instructions to swapping the MPLS labels are injected to the inner routers. Before the packet leaves the MPLS network, the MPLS label is removed via the pop label action [72].

HN

Deploying MPLS in hybrid network is straightforward if the network administrator chooses the SDN switches to create the MPLS. The tunneling is then created via those switches using the technique presented in the previous section. However, when the administrator is forced to use both SDN and legacy switches to form the MPLS network, either because the number of available SDN switches is not sufficient or because legacy switches are placed in strategic locations, MPLS will require a hybrid control. SDN controller manages SDN switches and an MPLS controller manages traditional routers. Once labels are issued, The SDN and MPLS controllers exchange labels and topology information for a smooth

MPLS operation.

Analysis

Even though MPLS was developed to simplify shortcoming of IP and ATM, the MPLS control plane remains very complex in traditional networks. SDN offers a central control that is exploited by MPLS to solve this issue. Combining a central controller and an MPLS extension to OF makes SDN a suitable platform for this protocol. With SDN, the controller knows the topology of the network and knows the location of the MPLS routers. It injects add MPLS labels rules in ingress routers, swap labels rules to internal routers, and pop the MPLS label rules to border routers. Hybrid networks can be favorable for MPLS only if the MPLS network spans the SDN switches only. If legacy routers are added to the MPLS network, the scheme becomes more complex than that of MPLS in traditional routers.

3.2.3 Multicast

Traditional networks

Multicast is a type of routing where a source sends a message only once but this latter is received by multiple destinations. Three entities characterize it: the address of the multicast group, the multicast tree and the tree creation based on a receiver joining a group. The multicast group IP address defines the recipient of the multicast message; hosts interested in joining a particular group receive packets destined to its IP address. At the same time, a source uses it as a destination address and sends the packet once as a unicast message, not being aware of the hosts that will receive it: it is the routers' responsibility to duplicate

and send the message along many interfaces to the different members of the group. Signaling between hosts and their gateways (routers) about joining or leaving multicast groups is managed by the Internet Group Management Protocol (IGMP). Whenever a host requests to join a group, the corresponding gateway router initiates the creation or modification of a multicast routing tree. This tree ensures that all members of a group receive the multicast messages as it indicates to routers when to duplicate and whom to forward to. Many protocols have been developed for the tree construction, such as the Protocol Independent Multicast (PIM) and the Distance Vector Multicast Routing Protocol (DVMRP) [73].

SDN

Many Multicast approaches in SDN were presented, mostly targeting SDN based datacenters. These approaches actually took advantage of the SDN architecture and control plane in order to address the challenges faced today with Multicasting. SDN simplifies Multicast routing since the controller possesses a global and comprehensive view of the network, which greatly decreases the overhead in forming multicast trees. The bottleneck for Multicasting in SDN is thus reduced to finding the most efficient algorithm for the tree creation. In fact, research in the field is mainly concentrated on that issue. For example, Scalar-pair Vectors Routing and Forwarding (SVRF) [74], a multicast routing and forwarding algorithm for SDN-based datacenters, is conceived as a solution for the problems incurred by multicast specifically in datacenters. SVRF is based on the Chinese Remainder Theorem and the prime number for membership calculation, and proved to have faster processing time and less memory usage when compared to the state of the art “Bloom Filter”. Nevertheless, the authors clearly state that SVRF works only in pure SDN networks. Avalanche Routing Algorithm

(AvRA) [75] is another proposed multicast routing algorithm for SDN. AvRA is designed to work for any Tree or FatTree topology in datacenters. Even though it is implemented as a controller module, it does not require any additions to the SDN switches. In fact, AvRA is implemented as an OpenDaylight module that creates and updates routing trees for Multicast groups in polynomial time. The module monitors the network for hosts sending IGMP messages to subscribe to the different groups via the “ListenDataPacket” of Opendaylight. Multiflow [76] is an SDN application that operates like Avalanche. It takes advantage of the exposed NOX APIs to employ IGMP snooping and determine which hosts are interested in joining multicast groups. Multiflow then runs the best route discovery to create the trees based on Dijkstra Algorithm.

HN

If one could expose the topology of the hybrid network to the controller, then the problem becomes similar to that of multicast in pure SDN. The multicast tree can be dynamically computed using techniques similar to those presented in the previous section. Communication between legacy and SDN switches uses an approach similar to that of fibbing [18]. The topology discovery is initiated by SDN switches which then share it with the controller.

Analysis

Multicast is by nature computationally expensive, because every time a host joins or leaves a group, multicast requires the creation of a modified multicast tree to assimilate the changes. This process is relatively demanding in traditional networks, which basically relies on flooding the network to discover the topology and prune the Multicast tree. With SDN, the full topology and the hosts requests are

available to the controller. Topology changes are transparent to the controller because their representation is handled by dedicated topology modules or apps. As discussed previously, SDN Multicast is henceforth simply a tree creation problem.

3.2.4 VxLAN

Traditional networks

VxLAN is an L2/L3 technique that enables two or many networks to be connected as if they are on the same L2 domain. It is especially useful in data centers for connecting Virtual Machines that reside on two disjoint networks. VxLAN introduces many advantages to data centers especially that it expands the number of possible VLANs. In fact, the length of the current VLAN identifier and the use of the Spanning Tree Protocol limit the amount virtualization. VxLAN provides easier network isolation for multi-tenant environments, since L2 isolation is limited by the maximum VLAN number and L3 isolation is limited by the fact that tenants might use the same internal IP addresses. VxLAN also reduces the size of MAC tables at the rack extremity switches [77]. VxLAN can only be implemented in a network which supports multicast, IGMP and PIM. The mac address of the destination VM needs to be known to establish a communication line. Assuming the case, the VxLAN Tunnel End Point (VTEP) appends the VxLAN Network Identifier (VNI) of the destination VM and encapsulates it in a VxLAN header to be sent to the corresponding receiving VTEP. This latter checks if the VNI/MAC association exists, and forwards the de-capsulated packet to the destination VM. If the MAC of destination is not available, the source VM sends a regular ARP message. This message is collected by the VTEP which adds the corresponding VxLAN header and encapsulates in a multicast packet sent to

the group associated with the destination VNI. Receiving VTEPs process packets and broadcasts it to nodes associated with the VNI. The destination VM then responds with a normal unicast ARP response, which is re-encapsulated by the receiving VTEP and sent back to the sending VTEP and ultimately the sending VM [78].

SDN

The VxLAN protocol extension is readily available in Open vSwitches. OVS provides VxLAN tunnels, implemented in kernel space and VxLAN encapsulation and decapsulation. The VxLAN tunnels are mapped to virtual ports of the switch, i.e. tunnels are created by adding a port of type VxLAN to the OVS. The overlays between the OVS are configured by assigning the IP of the other end of the tunnel to that port and the VNI to the tunnel. OVS also allows specifying the OF port of the tunnel, which provides better control over the interfaces in the network. Flow entries loaded to the switch by the controller direct traffic for the VMs residing on both sides of the tunnel end points. In case of a missing forwarding rule, OVS sends only the first packet of the flow to the controller and handles consequent flow packets using the same rule, in order to accelerate traffic flow [[79], [80]].

HN

VxLAN can operate in hybrid environments similarly to other tunneling protocols such as MPLS. As long as the VTEP are OVS, the protocol operates exactly the way it does in pure SDN networks. As the OVS switch is programmed to create a VxLAN tunnel on a given OF port, which is then tied to a remote OVS destination, the tunnel is created regardless of the nature of other switches in the

network

Analysis

Based on the previous sections, it is better to adopt SDN for the VxLAN protocol for many reasons. First of all, SDN can avoid ARP packets flooding and ARP responses for the MAC address inquiry at each VxLAN connection. ARP resolution can be performed at the SDN control layer, and directly provided to the VMs. Also, SDN allows control over the VxLAN ports and the corresponding VNI for better traceability. Finally, support for VxLAN already exists in OVS, which means that there is no incurred overhead.

3.2.5 QoS

Traditional networks

All services cited above were conceived to improve the performance over networks. With the emergence of multimedia applications, another parameter developed, aiming to deliver these demanding services in the best quality possible. Quality of Service (QoS) aims at guaranteeing the quality required for a good user experience. It basically obeys four constraints: reliability, jitter, delay and bandwidth. QoS is achieved by many techniques that address each one or more of the four parameters. Over provisioning the service with bandwidth or buffer space might seem the easiest approach; however it is an expensive approach. Another technique deployed at the receiving side consists in using buffers before delivery. The buffer then delivers packets to receiver at a constant rate, which smooths out jitters. Nonetheless, it does not help reduce delays. Traffic shaping is also considered for QoS, whereby transmission rate is regulated at the sender rather

than at the receiver. It is achieved by an agreement between sender and the actual network (links): a service level agreement, which states the traffic capacity supported, is exchanged between the sender and the carrier at the start of each connection. If both parties keep their side of the agreement, the data will be delivered smoothly and congestion avoided [81].

SDN

SDN is expected to assist in delivering QoS, consequently much research arose in this area. OpenQoS [82], an OpenFlow controller which is dedicated to ensuring quality for multimedia applications, separates traffic into multimedia and regular flows. OpenQoS then routes each type of traffic according to algorithms that ensures guaranteed routing for multimedia and normal shortest path for regular data traffic. QoSFlow [83] provides traffic shaping and congestion avoidance by controlling scheduling algorithms and packet schedulers, in the aim of improving QoS. It extends OF 1.0 by creating a new datapath. This latter is a combination of the traditional OF datapath and a QoS module composed of three elements: Traffic Shaping, enqueueing of flows and Packet Schedulers. Traffic shaping and packet schedulers manage the resources and the traffic available to the queues according to the controller's QoS commands. This technique delivers high flexibility and on-demand resources provisioning for applications. Q-Ctrl [84] is another proposal for a QoS framework in SDN. It realizes end-to-end QoS for multimedia applications, but also for scientific and web applications. Q-Ctrl sits on top of the floodlight controller. It is composed of a QoS manager, network topology monitor and database, and a QoS flow injector. Applications send their requirements in quality to the QoS manager, which manages network resources based on inputs from the network topology monitor. The Flow injector then

inserts flows into the SDN controller and subsequently the OF switches.

HN

QoS can be envisioned in hybrid networks. If the QoS module in (or on top of) the SDN controller exposes APIs to “sensors” on the network, it can collect information such as available resources and link traffic. This information is then presented to a QoS module that manages resources according to the demand.

Analysis

QoS is also a service that is better deployed in SDN. In fact, a QoS module in SDN has access to all sorts of data in real time. This obviously enhances the quality of the services especially that the manager can regulate throughput and bandwidth on the fly. It monitors the virtual and physical network, constantly checks for the applications requirements, and regulates resources to requirements as they vary.

The different services and their implementation are summarized in table 3.1.

3.3 Consequences on Network Management

Hybrid network introduced heterogeneity in the control planes, which became a major problem to consider in order to avoid inconsistencies in the network states. Another dimension to consider in addition to network services is the network management (NM) plane, which ought to change to support this evolution. Accordingly, the Open Networking Foundation (ONF) [85] emphasized the importance of NM and added a management layer to the SDN architecture,

	Traditional	SDN	Hybrid
Firewall	One firewall is simple to implement	Implementing distributed firewalls is more complex SDN allows distributed firewalls	Legacy firewalls can be employed in SDN networks
Proxy	Simple	Simple	Simple
DPI	Complex	Centralized architecture not favorable SDN does not give an edge to DPI	Take advantage of traditional DPI systems and integrate them in SDN
IDS/IPS	Complex	SDN gives an edge to IPS although it does not improve performance of IDS compared to traditional networks	Legacy IDS/IPS can be implemented in SDN although the performance of SDN-IPS is better
MPLS	Very Complex	SDN is a good architecture for MPLS	Easy if MPLS is deployed on SDN switches, very complex otherwise
Multicast	Very Complex	Easy thanks to SDN control plane	No added value beyond that of pure SDN
VxLAN	Very Complex	Easy thanks to SDN control plane	Easy if tunnel end points are SDN switches, complex otherwise
QoS	Complex	Better performance thanks to the real time reactivity of SDN	Feasible but no added value beyond that of pure SDN

Table 3.1: Network services in traditional networks, SDN and hybrid networks

indicating that if NM is not given enough care to be able to support SDN, this latter would not function properly [9]. In a later issue [86], the ONF introduced Network Orchestration, which exploits the network virtualization offered with SDN, abstracts physical resources, and adaptively allocate them to network services [87]. Orchestration binds the gap between requirements of applications and network operators on one side, and the available network infrastructure on the other [88]. As such, NM becomes an important player for network orchestration and SDN exploitation. Consequently, we investigated NM with SDN, as a first step to defining how management of hybrid networks can be designed. Additionally, we further elaborated on the Fault, Configuration, Accounting, Performance, Security (FCAPS) model and study its applicability to SDN, and proposed the categorization of network functions as a guideline to implement SDN management.

In [89], Wickboldt et al. present an overview of the development of SDN throughout the years, starting from the year 2008, when OpenFlow was first introduced, up to the current status of the concept in academia and the market. However, the authors were particularly interested in management of SDN networks; in fact, as the whole network is shifting, so should its management. Obviously, traditional network management does not apply to SDN, since the requirement for the new paradigm is different than those of the traditional 7-layered networks. The authors compare management activities such as bootstrapping, resilience and availability activities, security, monitoring, network programmability and performance between the two schemes. The work concludes with a list of challenges that management faces with the current setting of SDN, and the authors argue that these challenges should soon be mitigated in order to avoid patching management add-ons in the future. Nonetheless, NM within SDN is

far more critical than it appears. In fact, as argued by Hall et al. [23], SDN was originally adopted in data centers, where trust was a minimal concern, since that environment usually featured only one control authority. However, in most networks, many authorities deploy their virtual networks on top of one shared physical fabric. In this case, the trust issue becomes a major challenge for SDN and more specifically for NM in SDN. The authors consider managing routing in such open and shared environments, in the aim of finding a solution for running private flows on the same physical switches. The work consisted of extending the Quagga routing suite with an SDN routing module. Internal routing is lifted off the routers up to the central control plane that keeps track of the state of the whole island (or Autonomous SDN network), with eBGP being used for routing across islands.

3.3.1 NM Layer for SDN

Many proposals considered management as an inherent part of the SDN controller and implemented it as a module in or on top of it, such as Frenetic [90] and Procera [91]. However, a major drawback of this kind of management technique is that the controller becomes a single point of failure; any error on the control plane may disable the controller and the management plane at once. The controller also becomes overloaded, especially if the network is large. Another drawback of delegating management functions to the controller is the programmable nature of that latter, which makes it prone to attacks and consequently, compromises the management plane [92].

Instead, researchers started considering other alternatives. The ONF started working on defining a scheme for the issue. The management layer was henceforth marked as a layer perpendicular to the three layers of SDN: data, control

and application, as shown in section 2.2 Figure 2.2. It is basically in charge of all functions that do not belong to any of the three layers, such as all client related tasks, resource allocation to users, identification and authorization tasks, bootstrapping and so on. This layer communicates with all three layers as shown, however no clear characterization of the functions of the management layer has been set yet. Furthermore in [93], Devlic et al. analyze the above model provided by ONF. In this model, the OF-Config protocol, set by the ONF, was used to configure and communicate with all three layers of SDN. The implementation of a special protocol dedicated for management (and different from OpenFlow) by the ONF implied that the separation between management and control is necessary. The setting consisted of virtualizing carrier networks and offering different control granularity to different users depending on their roles and privileges. The authors were particularly interested in defining the roles of OpenFlow and OF-Config in management, as currently presented by the ONF. An experiment was carried out by mimicking the steps of device configuration in the use-case, and results showed that this model features many weaknesses: first of all, the OF-Config points and the controllers are considered as distinct units in the ONF model. However, these two should be tightly connected in order to ensure fast reaction within the system. Moreover, the authors detect gaps in OF-Config and OpenFlow protocols, especially when it comes to the discovery of physical resources, configuration of logical links, and instantiation of logical switches.

Whether control and management are merged together within the controller or they are separated into two different entities, both ideas were subject to further research; different models were proposed, each with its own advantages and disadvantages.

3.3.2 Proposed Management Schemes

The management schemes presented in the literature are very diverse in their nature: some researchers proposed a controller-manager, others thought of distinct policy based manager, and some even considered of distributed management. The different models and their applications are reviewed in this section.

Management as an Inherent Part of the Controller

As a first attempt to model network management in SDN, Wang and Matta reviewed proposed SDN controllers and the corresponding network architectures in [94], focusing solely on management in the control plane; in other words, the authors considered the management layer to be the control layer itself. The survey aimed at identifying the management schemes for different controllers. Starting with NOX SDN controller, the management layer featured only one controller, which might not be efficient for large expanding networks. Another SDN controller, Onix, on the other hand, creates many instances of the controller, each allocated to a certain subset of switches and routers, which better serves scalability. Network virtualization offered by FlowVisor responds better to efficient resource utilization by exploiting network virtualization; it permits different NOX controllers residing on top of FlowVisor, to share the same network substrate via virtual partitioning and isolation of the network. Another important aspect of management is the ease with which an operator can control the network, a characteristic provided by PANE. Similar to NOX, PANE has one central controller in the management layer, however PANE offers an API that allows users to have control over the network, such as for reserving bandwidth. Nevertheless, a major drawback is the lack of full support for QoS. Again, the authors concluded their work by indicating weaknesses of the surveyed SDN controllers' manage-

ment since these schemes are closely tied to TCP/IP and inherit its problems of mobility, QoS support, and security. Building on these drawbacks, they present the Recursive Inter-Network Architecture (RINA), a policy-based management layer that resides at the top of the SDN architectures and mitigates the above cited problems.

Policy-based Management

In [95], Smith et al. extend their previous work on management for traditional networks in [96] to be applied to SDN networks. They develop a resilience scheme centered on policy-based management that can reside on top of the SDN infrastructure. A resilience manager reads event measurements, monitoring data (bandwidth) and/or any fault or anomaly (attacks or intrusions) from the network, and chooses the appropriate action to execute based on predefined management patterns. These patterns are high-level formulations of the management policies that express the corresponding configuration to apply in response to the occurrence of a given event. The patterns or rules are associated to roles offline, but the applications are associated to roles online, depending on the availability of resources. Examples of roles are: Rate Limiter, Traffic Classifier, Virtual Machine Replicator, Link Monitor etc. The framework was tested on Mininet using the POX controller. The framework was able to mitigate a denial of service attack 20 seconds after its start. Kim et al. elaborated on Procera [91], also a policy-based control framework. Procera provides a platform for network operators to easily control the SDN network via policies and rules. These high-level rules are then translated into OpenFlow like instructions, which are in turn implemented onto the forwarding plane via the controller. An event source is any network element (e.g. middlebox) that transmits any kind of information

to the policy engine. This latter processes the events and the policies installed, synthesizes the appropriate action to be taken, and sends it to the controller to be enforced on the fabric with OpenFlow. Procera aims at simplifying the deployment of complex policies on big dynamic networks, which was verified when tested on the Georgia Tech campus. Procera also proved efficient when tested in a home network. Han et al. revisited policy-based SDN management in [97], but from a different perspective: a framework called layered policy management (LPM). The paper addresses policy deployment on all three layers: application, control, and data. In fact, each layer features a set of challenges that are best solved locally; consequently three types of policy management techniques were developed: management for inter-module dependency (at the controller level), for inter-application dependency (at the application level) and for intra-table dependency (at the switch level). Policy segmentation was employed to resolve the first type of dependencies, whereas the second type was solved by allocating priority scores to applications. Finally, “flow isolation”, which consists of appending tags to every new flow in the table, was used to settle intra-table dependencies. The authors validated their work via a proof-of-concept implementation on Mininet with Floodlight, and tested on a configuration taken from a real backbone network in Stanford [98].

Programmable Management

In [92], Kukliński introduced the programmable SDN manager, a new perspective of SDN management. The author argued that since the whole SDN network became programmable by software, so should its management. Delegating management functions to the controller itself shows many shortcomings that could be mitigated by adding a centralized management node to SDN. In this scheme

some management functions could be delegated to the controller, especially those related to applications that are installed and running on the controller, but the presence of the SDN node implementing large-scale management functions and looking over the whole network assures more robustness. However, this scheme still features a single point of failure and a bottleneck (the SDN management node), which the author proposed to overcome by adding programmability to the management, distributed on all levels: manager, controllers and switches. This method obviously incurs changes to the SDN architecture as currently defined and to the switches as well, but proved to cope better with scalability, response delay and reliability. While network management has proven to be a hard task in traditional networks, it is not greatly simplified with SDN. In fact, while SDN alleviates the burden of configuring each network element separately, the controllers as they are currently designed provide limited management interfaces. For example in [99], the authors developed “Semantic Network Management” aiming to present an easy, high-level management to operators, who will only have to define high-level information, rather than subnets, IP addresses, and routes. The 32-bit IP address space is divided into Group pool and ID, and Host pool and ID, which are later used to simplify forwarding and reduce entries in flow tables. For example, the method allows to aggregate flow entries for hosts belonging to the same group, and consequently, it automatically assigns addresses such that the number of flow entries in tables is minimized. Semantic Network Management was implemented in EasyWay, an application built on top of the RUNOS OpenFlow controller. Benefits of the presented approach span performance and flow tables learning, while the overhead is kept minimal.

Management Transition: Legacy networks to SDN

In [100], Kuklinski and Chemouil exposed the challenges of SDN network management by comparing it to that of traditional networks. Starting with the FCAPS model, they showed the different functionalities and drawbacks of OpenFlow, mainly its inability to perform controller programmability, a new management function required in SDN. Similarly, [101] pointed out that even though SDN presents an opportunity to simplify management by providing a central data acquisition/control entity, it also creates additional requirements related to network programmability, dynamism in policy creation, and network bootstrapping. This latter was invoked in [102], where InitSDN was introduced as a tool to manage network bootstrapping on two levels: the data slice and the control slice.

3.3.3 Guidelines for a Management Framework

Even though the controller can carry out the role of a manager, such a choice might have been carried out for practicality only, especially that management issues in SDN have only recently been considered and identified. In fact, relying on the controller solely might overload it especially that it is already in charge of controlling network services. Consequently, choosing the right type and amount of management functions to be consigned to the controller becomes crucial. The separation between SDN manager and SDN controller is recently an active area of research; many papers tackle this issue and expose the different related problems [100]. Building on the literature and on the FCAPS model, we define its applicability on the changing management environment with SDN, given the roles of the manager and controller.

Level	Manager	Controller
Fault		X
Configuration	X	X
Accounting	X	
Performance	X	X
Security		X

Table 3.2: Proposed FCAPS allocation in SDN

Extension of FCAPS to SDN: Categorizing the Functions

FCAPS is an ISO standard for telecommunications and network management, and the most widely used [103]. We categorize the different network functions into the corresponding areas of FCAPS and SDN level (controller or manager). Each level can be allocated to either the controller or the manager depending on the function that it encompasses (Table 3.2). We define a rule of thumb which is in-line with what is proposed in the literature [100], and with the separation that is observed in the OpenDaylight SDN controller [104], as follows: *Critical tasks that need to be handled in real-time or for an extremely short duration are assigned to the controller, whereas tasks that require monitoring over a larger time scale are assigned to the manager.*

Accordingly, Fault and Security are attributed to the controller due to their critical impact. Configuration, Accounting and Performance management require rather longer processing and data analysis time; they can be thus attributed to the manager. However, some functions related to dynamic configuration or performance management, such as traffic engineering (TE), should be handled quickly and thus should be assigned to the controller instead. These are marked in grey in Table 3.2.

Figure 3.2 lays down a subset of the main management functions with respect to their relation to Fault, Configuration, Accounting, Performance or Security.

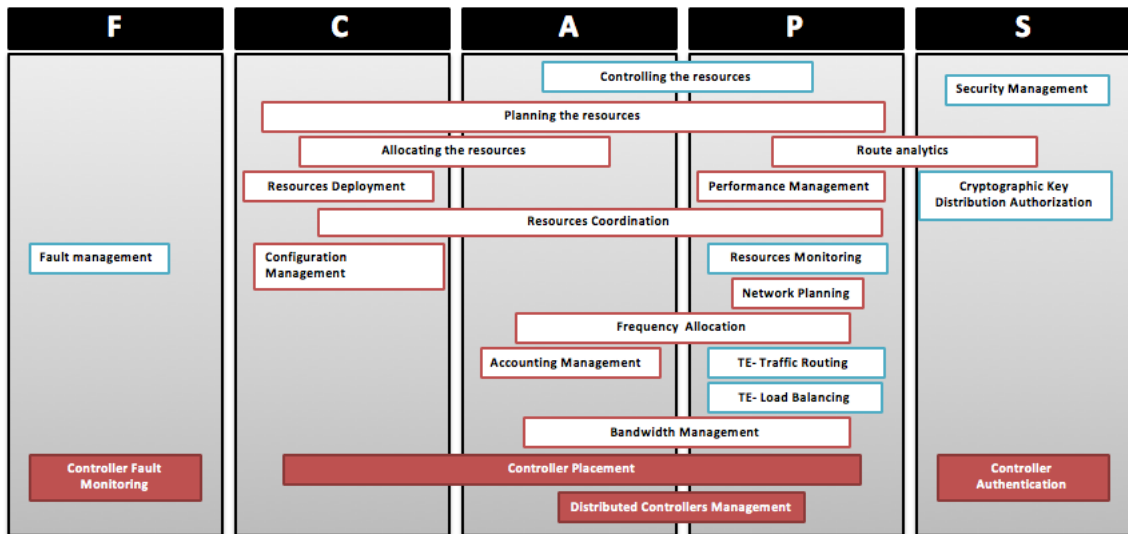


Figure 3.2: Map of management functions for building management schemes for SDN

Each function is assigned to either controller (blue border) or manager (red border). SDN-specific management functions are shown in red rectangles. Another guideline for the classification can be followed based on proximity and the extent of knowledge that each entity has regarding the different layers: *Tasks that are carried out at layers 2 and 3 of the network are assigned to the controller, whereas tasks that are carried out at layer 4 and above are assigned to the manager.*

Proposed Architecture

The choice for the rule of thumb to employ for the allocation of management applications depends on the type of network and the nature of the services running on top. All functions allocated to the controller, defined as *Management Modules*, are implemented as modules of that controller, having the same level of importance as control modules, for example when it comes to reserving resources. However, they use OF-Config to communicate with the network substrate instead of OpenFlow. Functions allocated to the manager, defined as *Management Ap-*

plications, will be implemented as applications on the application layer. Their role is less critical than that of *Management Modules*, which justifies the fact that they are implemented further away from the data layer. At the same time, their proximity to the top layers allows them to run layer 4-and-above management functions more efficiently. These applications do not communicate with the devices; instead they acquire the needed statistics and infrastructure information indirectly from the controller via Northbound Interface. Recent research started considering the switches' role in undertaking some control tasks to ensure scalability of SDN networks [92], [17], [105]. Even though this could be controversial given the SDN specific control/data planes split, it does allow switches to contribute to the management plane. In this case, the switches can help management applications in data collection or policy enforcement, thus offloading these functions from the controller. Consequently, the *Management Applications* would be able to bypass the controller and directly communicate with the switches. However special care should be taken in order to ensure consistency between controller and manager.

Accordingly, the proposed architecture is a virtual management plane that is perpendicular to the layers of the SDN, and intersects the control and the application layers (Figure 3.3). The Orchestrator encompasses both manager and controller.

3.3.4 POC: System Model and Implementation

To validate our framework, we modeled the different network entities and defined the performance cost as follows:

- Vector $A = \alpha_i$ contains the different applications α_i , and length of A is the

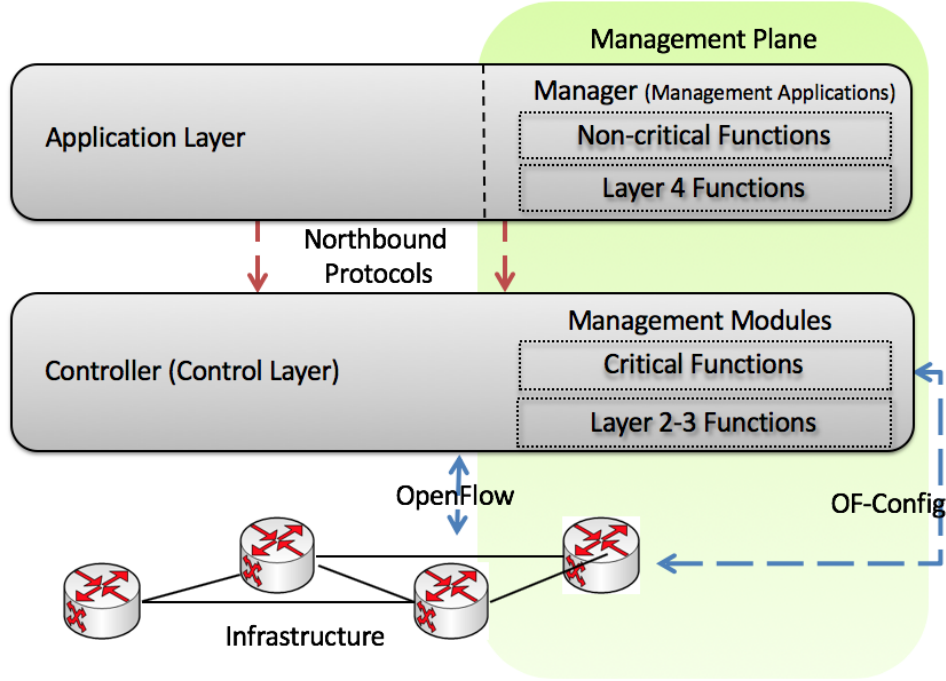


Figure 3.3: Proposed management framework based on management functions categorization

number of applications.

- Vector $\mathbf{R} = r_1, r_2, \dots, r_i$, where r_i is the performance requirement of application α_i .
- Vector $\mathbf{X} = x_1, x_2, \dots, x_i$, where x_i indicates if application is assigned to controller ($x_i = 1$) or to manager ($x_i = 0$). For scenario 1, \mathbf{X} is all ones, and for scenario 2, \mathbf{X} is all zeros.
- Vector $\mathbf{Y} = y_1, y_2, \dots, y_i$, where y_i shows the criticality of application α_i .
- α_c and α_m represents the overhead of assigning non-critical apps to controller and critical apps to manager, respectively.
- Vectors $\phi_c = \phi_{c,1}, \phi_{c,2}, \dots, \phi_{c,i}$ and $\phi_m = \phi_{m,1}, \phi_{m,2}, \dots, \phi_{m,i}$ show the

bad-assignment penalty of application α_i for controller and manager, respectively.

The bottleneck of the controller layer is its capacity (C), because an overloaded controller can cause a network downtime. Alternatively, the bottleneck of the manager is rather the communication bandwidth (M) to the controller (through northbound interface). The performance cost involves two sub-costs: processing delay cost from an overloaded controller and communication delay cost from running time critical apps on manager. To compute the processing delay from the controller, we take the multiplication of the performance requirements of applications with the controller penalty of bad-assignment, summed over all apps, and then divided by the controller capacity. Alternatively, to compute the communication delay cost, we take the multiplication of performance requirement of applications with the manager penalty of bad-assignment, summed over all apps, and then divided over the communication bandwidth. Consequently, the performance cost Θ is:

$$\Theta = \sum_i^A \frac{(1 + \phi_{c,i})(\alpha_i r_i)}{C} + \frac{(1 + \phi_{m,i})(\alpha_i r_i)}{M} \quad (3.1)$$

where

$$\phi_{c,i} = \alpha_c (Y_i \otimes X_i)$$

and

$$\phi_{m,i} = \alpha_m (Y_i \otimes X_i)$$

We simulated the system using MATLAB. We measured performance cost in three scenarios as shown in Table 3.3. The application criticality vector Y was initialized randomly so that 50% of apps are critical and 50% are non-critical

Scenarios	
Scenario 1:Proposed Framework	Distribution of management apps among manager and controller
Scenario 2: Controller Only	All management applications running on the controller
Scenario 3: Manager Only	All management apps are running on the manager

Table 3.3: Simulation scenarios

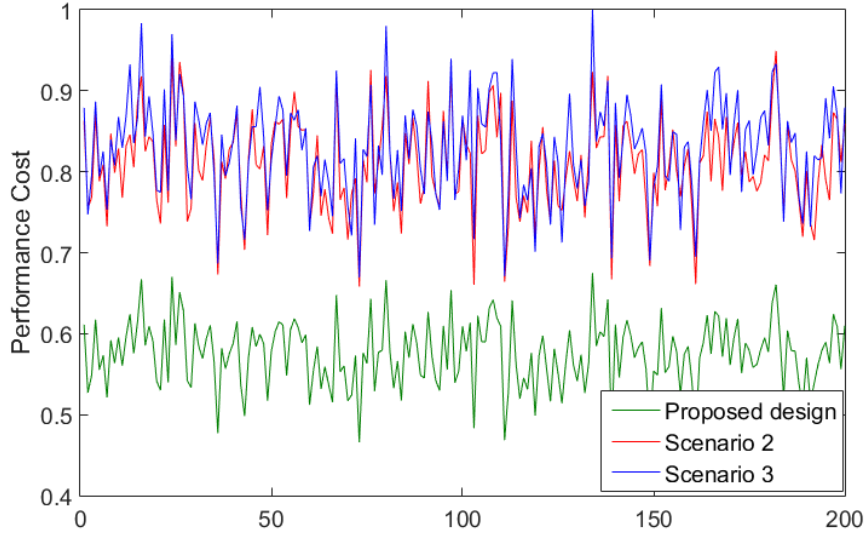


Figure 3.4: Comparison of the normalized performance cost of the 3 scenarios

to ensure fair allocation cost between manager and controller. The controller capacity and communication bandwidth are initially set to half of their maximum capacity (50% on a normalized scale).

Results and Analysis

Figure 3.4 shows the performance of the 3 scenarios with network operation. The proposed model outperforms the other two scenarios in normal network operation where application requests vary over time. We can see an improvement of around 27.5% between the proposed design and the two other scenarios. Figure 3.5 shows the performance cost when controller capacity increases while the communication BW from manager to controller is kept at 50% of its maximum value.

When controller capacity is at 0, the performance cost shows an asymptotic behavior towards infinity in scenario 2. This is expected because if the controller capacity is close to 0, then no app can run on it. The proposed scenario also shows high performance cost at 0 due to the contribution of the controller cost term in the proposed system's cost function. Scenario 3 has a constant cost due to the fact that the manager is not affected by controller capacity. When the controller capacity increases (Figure 3.6), performances of scenario 1 and the proposed system decrease, because C starts reducing the controller cost term in both scenarios, to the point where the proposed system starts showing better performance than scenario 3. As controller capacity keeps increasing, the cost of scenario 2 becomes equal and then less than scenario 3, and asymptotically it becomes superposed to the cost of the proposed design. This is also expected because for a big capacity, the processing bottleneck of the bad allocation of non-critical tasks to the controller disappears and thus its performance cost becomes comparable to the ideal distribution of tasks. A similar analysis can be performed by varying communication delays between manager and controller/ infrastructure (Figure 3.7). When the delay is small (i.e. the bandwidth M is high), scenario 3 shows a behavior comparable to the proposed design because under this condition, there is no overhead incurred from the bad allocation of critical tasks to manager. As the delay increases, the performance cost of scenario 3 increases and surpasses the performance cost of the proposed design. For high delays the cost of scenario 3 increases. This also affects the cost of the proposed design due to the contribution of the manager cost term in its cost function. The cost of scenario 2 is nearly constant, because the controller cost is not affected by the bandwidth capacity.

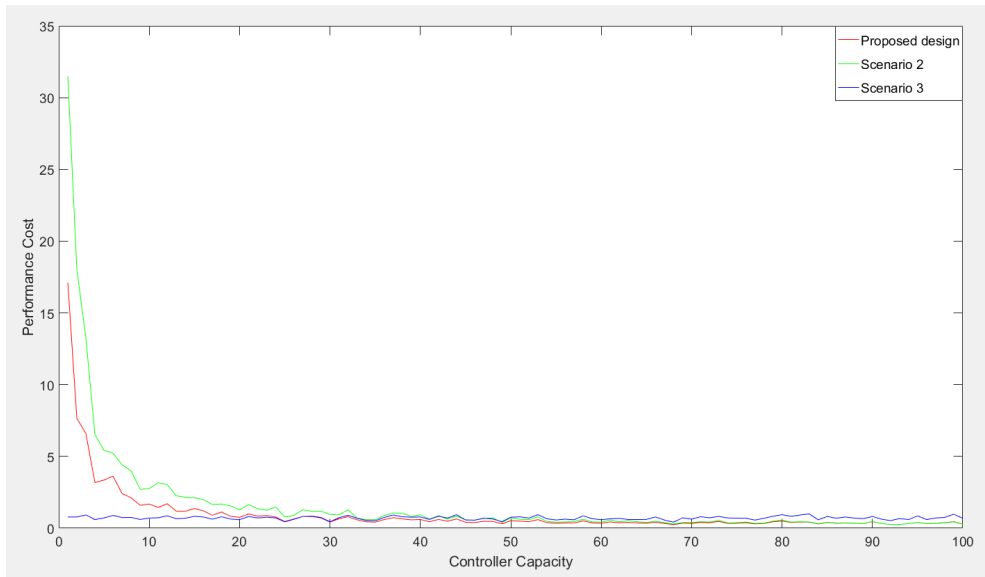


Figure 3.5: Performance cost with varying controller capacity

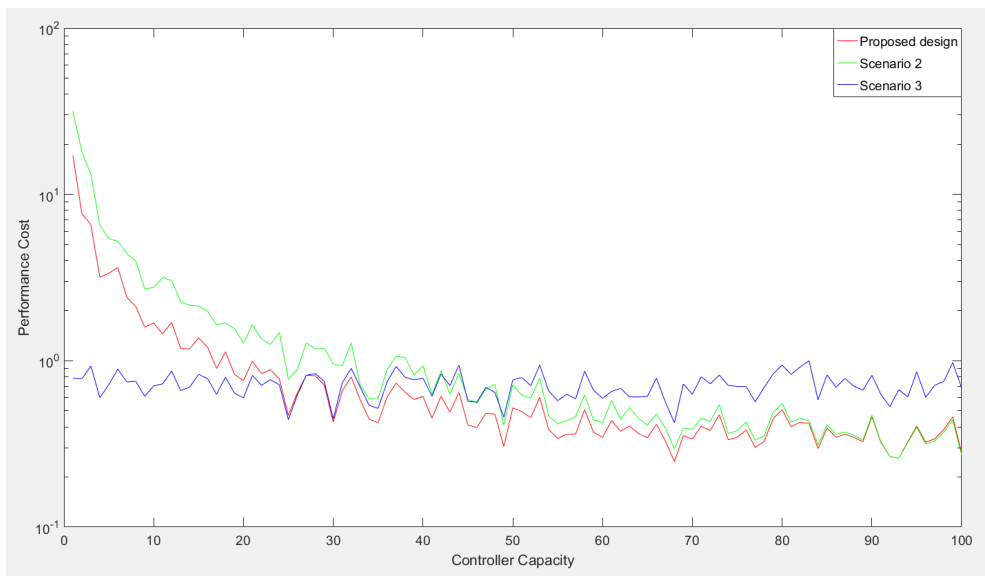


Figure 3.6: Performance cost with varying controller capacity in semi-log scale

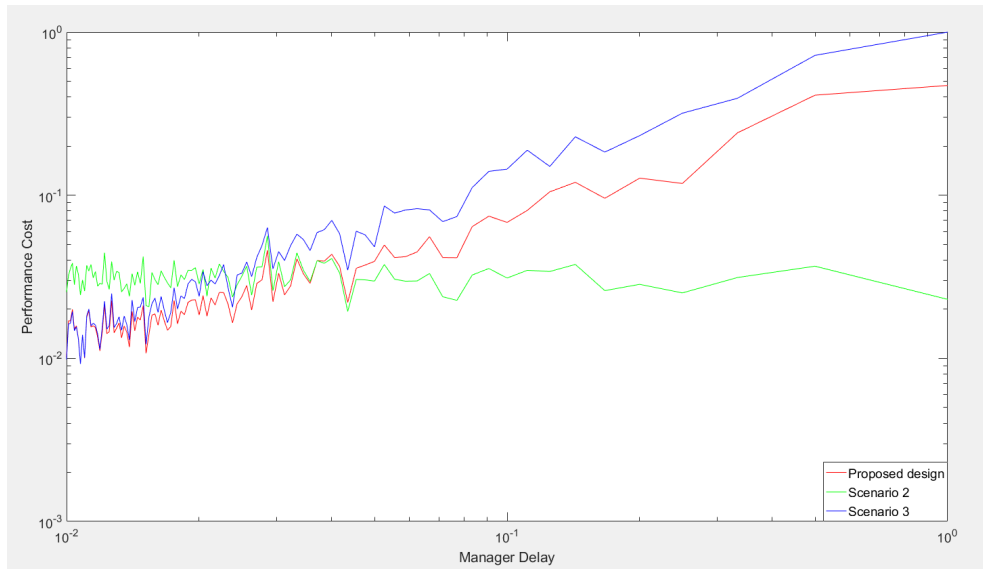


Figure 3.7: Performance cost with varying Delay

Chapter 4

Performance Modeling and Testing of Centralisation (SDN) Versus Decentralisation (OSPF)

4.1 Network Convergence in SDN Versus OSPF Networks

Software-Defined Networking (SDN) brought about several benefits in terms of flexibility and ease of management. SDN also promised easier network convergence [1], especially that the SDN architecture introduces a single node to supervise, control, and manage the network: the controller. This latter has full view and knowledge of the network; it can access and configure all resources, and quickly react to any changes that occur in the links or devices. Intuitively, one would think that the centralization introduced with SDN does not only provide easier convergence to a new state, but also a faster one when compared to

distributed routing protocols. In fact, in SDN, the controller is the only node responsible for making routing decisions. Alternatively, in legacy IP networks, all routers exchange topology information in order to agree on the state of the network; subsequently each node creates its own routing table. However, some research showed that SDN convergence is not as fast as anticipated, and that some unseen delays that hinder the expected fast convergence also exist in SDN-enabled networks [106]. Moreover, distributed routing protocols have been developed for years: convergence, being an important aspect of a functional network, was thought of early on, and techniques to improve it were introduced. For example, incremental Shortest-Path First (SPF) were developed to increase convergence efficiency [2]. In this section, we are interested in theoretically comparing convergence of centralized SDN with the standard OSPF routing protocol and then, experimentally testing the two network types and evaluating their convergence speed, to verify that the theoretical study matches the actual results. Subsequently, we analyze and try to identify the causes behind the mismatch between what was theoretically sound and the experimental implementation.

4.1.1 Network convergence

Related work

Network convergence was always a concern, given the importance of failure recovery and fast reaction to topology changes, for limiting lost traffic and interruption of services. Many papers compared the convergence of intra-domain routing protocols [107, 108] with the aim of finding the protocol that delivers the fastest convergence, and concluded that OSPF has faster convergence than RIP and EIRP. Pei et al. [109] model the convergence of path vector routing protocols

such as BGP to analyze their convergence delays with respect to topologies and network sizes. With the introduction of SDN, new interest in investigating convergence delays of this new technology as compared to those of traditional routing protocols surfaced. In fact, intuitively the characteristics of SDN should deliver fast, if not seamless network recovery, after a topology change. Zang et al. [106] investigated this issue. They modeled the convergence process in OSPF and SDN at a high level, and experimentally tested both networks to determine the delay it takes for each network to converge. Their results showed that OSPF performs better than SDN. However, when the link delay increases, SDN starts showing faster network convergence. In fact, in SDN, the controller is the one responsible for topology discovery and path computation. Consequently, advertisements are not flooded throughout the network, and thus the speed of the links does not affect convergence. Alternatively, convergence delay in SDN is affected by the performance of the controller, and by the efficiency of the routing modules used. Although one would expect SDN to deliver faster failover, the results in [106] are not surprising, especially that the OSPF protocol has been thoroughly studied throughout the years and continuously enhanced to deliver faster convergence [110]. SDN routing performance was also compared to inter-domain protocols. In [111], Gopi et al. compared the convergence speed of SDN to that of BGP in networks of different sizes. Results showed that SDN was able to deliver faster convergence, especially as the network grows in size.

Convergence General Definition

In traditional networks, network convergence is defined as a network state in which all routers have the same routing information, and their routing tables contain paths to all destinations in the network [112]. Alternatively, in SDN,

convergence can be defined as the state of a network where all destinations are reachable. The definition of convergence is slightly different in SDN; it is defined as the state when the controller has an updated map of the network with shortest paths to all destinations. Consequently, the states of the flow tables in the switches become associated to whether the controller dispatches the rules to the switches or not. Even though network elements have different roles in each network type, the main convergence aspect is similar in both schemes. Generally, convergence takes place in four steps [113]:

1. Identifying the topology event. The event can be any change related to topology; however, we are mainly focusing on link failure due to its impact on network operation and availability.
2. Transmitting the event. This step concerns propagating the event to the appropriate network nodes. In traditional networks the nodes are routers, whereas in SDN, the concerned network node is the controller.
3. Handling the event. This is achieved when the nodes restore the converged status of the network, by calculating alternative paths.
4. Updating the routing and forwarding tables.

These high-level steps are translated differently in traditional and SDN networks, given the different characteristics of each technology.

Convergence in Traditional Networks

We chose OSPF as an example to analyze convergence in traditional networks, because it is a standard that has good convergence performance compared to other distributed protocols. Topology changes are detected in OSPF in two manners: either through an indication from the physical layer (e.g. a “lost carrier”

message), or via the expiry of the “dead interval” delay, which corresponds to four times the “hello interval” [114]. When the topology change is signaled, the router which detects it sends an LSA update message to its neighbors, which successively ripple the message throughout the network. Each router then re-computes its routing table, and sends the update. Upon receiving the new updates, routers in turn compute their new routing tables. To control the table re-computation and avoid multiple SPF computations at each update receipt, the SPF computation at each router is held-off until an SPF counter is expired. Once all the routers in the network are notified of the topology changes and have computed their new routing tables, the network has converged.

Convergence in SDN

The convergence process in SDN is different from the above. Any change in SDN network topology is unswervingly indicated to the controller via a Port Status message sent from the concerned switch [72]. The controller updates its topology map, calculates shortest paths to all destinations, and sends new rules to the switches. The controller should not need to do network discovery again at each port-down status update. Only the map of the network topology is updated with the new event, assuming that if anything else had changed in the network, the controller would have been notified. This procedure is theoretical and its applicability depends on the implementations of the routing modules. Some controller implementations might do full network discovery after a topology change event is received, to ensure that the controller has the most recent topology of the network, which adds extra delays to the convergence process in SDN. Upon receipt of the new rules, the switches update their flow tables.

4.1.2 Theoretical Model

In this section, we model the network convergence in traditional networks and in SDN, to analytically compare convergence delays in these two types of networks.

Global Parameters

Some network related parameters are common to the two models. We define them as follows.

- The number of nodes in network (n)
- The number of links in network (l). Given that the network is a connected graph, it must have at least $(n-1)$ links and at most $\frac{n(n-1)}{2}$ links [115].

Consequently:

$$(n - 1) \leq l \leq \frac{n(n - 1)}{2}$$

- The depth of the network (f). We model the depth of the network from fault location to extremities as a n -vertex forest. It has an average tree-depth of $O(\log n)$ [116]. For the network in Figure 4.1 for example, the longest path in the network is 15 nodes. The best-case scenario occurs when the fault appears in the middle of the network, which makes it at most 8 hops away from extremities. If the fault is closer to extremities, then it is farther away from the extremity on the other side. Consequently, for the network in Figure 4.1, f varies between 8 and 15.
- The detection delay d_d or the time it takes to detect the change in topology. This delay is very small in SDN because it corresponds to a port status change at the SDN switches, which is directly sent to controller. In OSPF, the delay is also very small if detected by a loss of carrier at layer 2, which is

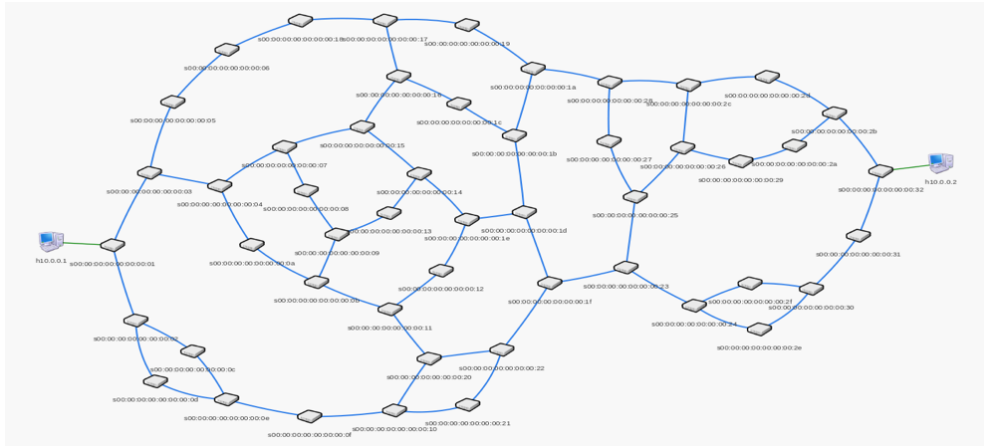


Figure 4.1: Simulated network of 50 nodes

more frequent than waiting for the “dead interval” to expire. In the latter case, the delay equals four times the OSPF “hello” interval.

- The SPF calculation time d_{SPF} . This delay is modeled by the ratio of the SPF instructions count over the instructions per second (IPS) of the network element performing the SPF calculation [117]. In our calculation, we used 2^{15} IPS. For the average case complexity, the SPF instruction count is of the order of $O(1 + n \log(n))$ [110].
- The congestion factor cf , which represents the network congestion at time t ; is modeled as a normal random variable drawn from the distribution $\mathcal{N}(1, 1)$.
- The link delay d_l , represents the delay observed on the links in the network.

All variables are summarized in Table 4.1.

OSPF Convergence Time

For the calculation of convergence time for OSPF, we define one more parameter:

Variables	
n	Size of the network
l	Number of links
f	Network depth at fault location
d_d	Detection delay
NLC	Network link capacity
d_{SPF}	SPF calculation time
IPS	Instructions per second
cf	Congestion factor
d_l	Link delay
d_{FOU}	Forwarding table update delay
CLF	Controller load factor
linkBW	Bandwidth of links between controller and switches
d_{CN}	Controller notification delay
d_{RF}	Rules dispatching delay
d_{FLU}	Flow table update delay

Table 4.1: Variables

- The forwarding table update delay d_{FOU} , which represents the time it takes to update the router forwarding table, on the order of 10's to 100's milliseconds [110].

As mentioned in section 4.1.1, the convergence delay in OSPF is the sum of the detection delay, the event propagation delay $d_l \times f$, the SPF calculation delay - which is accounted for twice, once at the first router, and once at the farthest router after having received all updates- and the forwarding table update delay. It is defined as follows (in this model, we assume that routers directly flood LSAs):

$$\text{ConvOSPF} = d_d + d_l \times f + d_{SPF} \times 2 + d_{FOU}$$

SDN Convergence Time

For SDN convergence modeling, we define additional SDN-specific parameters as follows:

- The controller load factor (CLF), represents the load of the controller at time t , and is modeled as a normal random variable drawn from the distribution $\mathcal{N}(1, 1)$. The controller is in normal load when the load factor is equal to 1.
- The controller notification delay d_{CN} is the time for the port status message to reach the controller. Given the high speed of links between controller and switches, this delay should be very small. In our design, it is modeled as the inverse of the bandwidth (linkBW) between controller and switches, which is defined as the product of the link bandwidth and the controller load factor at time t . The propagation delay between controller and switches is assumed to be negligible by design. Consequently:

$$d_{\text{CN}} = \frac{1}{(\text{LinkBW} \times \text{CLF})}$$

- The rules dispatching delay d_{RF} , is the time it takes for the controller to dispatch all rules to the switches. It is defined as the “Flow Mod” processing time (fabrication and transmission), and propagation time of the messages to the switches. The system can be modeled as a $M^n/M/1$ bulk arrival queue where n is the number of nodes in the network [118]. We assume that rules for a single switch are processed together and thus the number of customers in the queue corresponds to the number of switches. Consequently, the processing time of one “Flow Mod” message follows the exponential distribution with parameter μ , where μ is the processing rate of the controller [119]. The total processing delay is given by $\frac{1}{\mu}$. The propagation delay is defined by the number of switches in the network multiplied by the controller notification delay. The dispatching delay is

$n = 50$	$d_{\text{FOU}} = 10 \text{ ms}$
$l = 64$	$\text{CLF} = \text{normrnd}(1,1)$
$f = 8$	$\text{linkBW} = 10^8 \text{ bps}$
$d_d = 1 \text{ ms}$	$d_{\text{FLU}} = 10 \text{ ms}$
$c = \text{normrnd}(1,1)$	$\text{NLC} = 10 \text{ Mbps}$
$\text{IPS} = 2^{15} \text{ instructions/sec}$	

Table 4.2: Simulation parameters for scenario 1

therefore given by:

$$d_{\text{RF}} = \left(\frac{1}{\mu} + n \times d_{\text{CN}} \right)$$

- The flow table update delay d_{FLU} , which represents the time it takes to update the switch flow table, assumed to be in the order of 10's to 100's msec as in the OSPF update case.

Referring to section 4.1.2, SDN convergence delay is calculated as the sum of detection delay, controller notification delay, the SPF calculation delay at the controller, the rules dispatching delay, and the flow tables update delay. It is defined as follows:

$$\text{ConvSDN} = d_d + d_{\text{CN}} + d_{\text{SPF}} + d_{\text{RF}} + d_{\text{FLU}}$$

4.1.3 Testing and Results

We implemented the models using MATLAB, and observed the convergence delay in three scenarios, described below. In the first scenario, we swept over link delay d_l from 1 to 10 milliseconds with a step of 0.001, and watched the effect on convergence delay, with the parameter values in Table 4.2.

The results depicted in Figure 4.2 show that SDN convergence is not affected by the network link delay, whereas OSPF convergence is drastically affected by

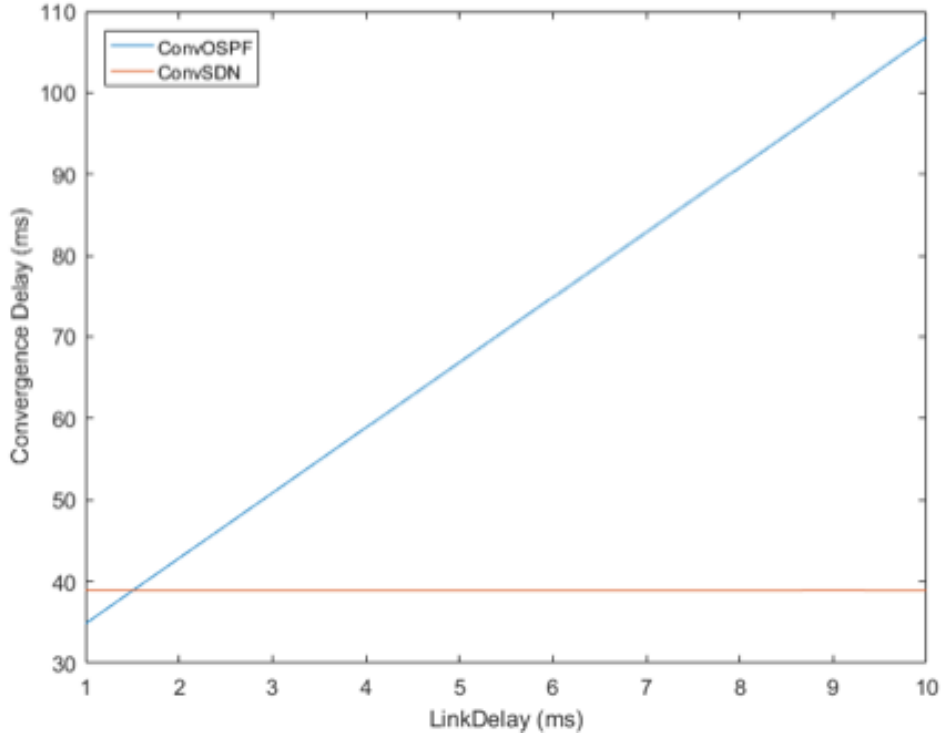


Figure 4.2: Effect of link delay on convergence delay

this delay. This is expected because convergence in OSPF relies on rippling OSPF advertisements throughout the network, and the speed of network links affect the time it takes for these messages to reach all the nodes.

Scenario 2 tests the system convergence for different values of the depth of the network at the fault point, i.e f is varied between 8 and 15, using the parameter values in Table 4.3.

Figure 4.3 shows the effect of the location of the fault on the convergence delay.

$n = 50$	$d_{FOU} = 10 \text{ ms}$
$l = 64$	$CLF = \text{normrnd}(1,1)$
$d_l = 1 \text{ ms}$	$\text{linkBW} = 10^8 \text{ bps}$
$d_a = 1 \text{ ms}$	$d_{FLU} = 10 \text{ ms}$
$c = \text{normrnd}(1,1)$	$NLC = 10 \text{ Mbps}$
$IPS = 2^{15} \text{ instructions/sec}$	

Table 4.3: Simulation parameters for scenario 2

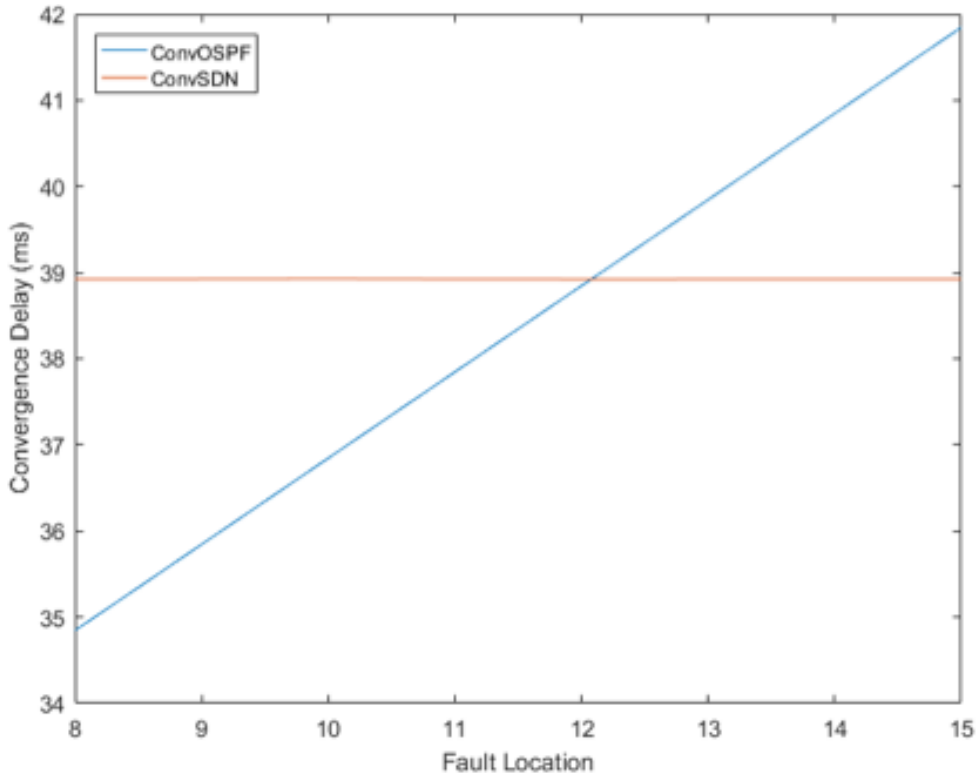


Figure 4.3: Effect of fault location on convergence delay

SDN is not affected by the fault location, as expected, because the controller is directly notified about the failure by the corresponding switch. On the other hand, OSPF is influenced by the failure location. In fact, when the fault occurs at node 8, in other words, when a link at the center of the network goes down, OSPF converges faster than when a link at the extremity of the network fails (fault location = 15). This is also expected because when a link closer to the edge fails, the advertisements should ripple through the whole network, whereas if the link in the middle fails, then, ads traverse nodes from both sides simultaneously, which saves time.

In the third scenario, we varied two parameters: we swept over network size n from 1 to 100 nodes, and over the number of links from $(n - 1)$ to $\lfloor \frac{n \times (n-1)}{2} \rfloor$ with a step of 1. The parameters were set as in Table 4.4

$d_l = 1$ ms	dFOU = 10 ms
$l = 64$	CLF = normrnd(1,1)
$f = 8$	linkBW = 10^8 bps
$d_d = 1$ ms	$d_{FLU} = 10$ ms
$c = \text{normrnd}(1,1)$	NLC = 10 Mbps
IPS = 2^{15} instructions/sec	

Table 4.4: Simulation parameters for scenario 3

	Nodes	#Hosts	#Links
Setting 1	50 OSPF routers	2	64
Setting 2	50 Open-Flow SDN switches	2	64

Table 4.5: Experiment settings

Convergence delay observed in each type of network is represented through a 3D plot, in Figures 4.4 and 4.5. We notice that OSPF converges faster than SDN when the network size and the number of links are small. However, as the number of nodes and/or the number of links increases, SDN shows faster convergence. The highest delays are observed when the network size and the number of links are maximum.

4.1.4 Simulation and Results

Network Setup

To experimentally test convergence delays, we created a network topology of 50 nodes in two configurations 1) OSPF network and 2) SDN network. The network topology is shown in Figure 4.1, and details of each configuration are shown in Table 4.5. We used Mininet [120] to implement the scenarios, because it provides an extension to Mininet hosts, which enables them to run the Quagga routing suite. These Quagga nodes are used as OSPF routers in scenario 1.

Open-Flow-enabled switches in scenario 2 are the normal switches used in the Mininet/Mininet emulators. For the SDN experiment, we used the Floodlight

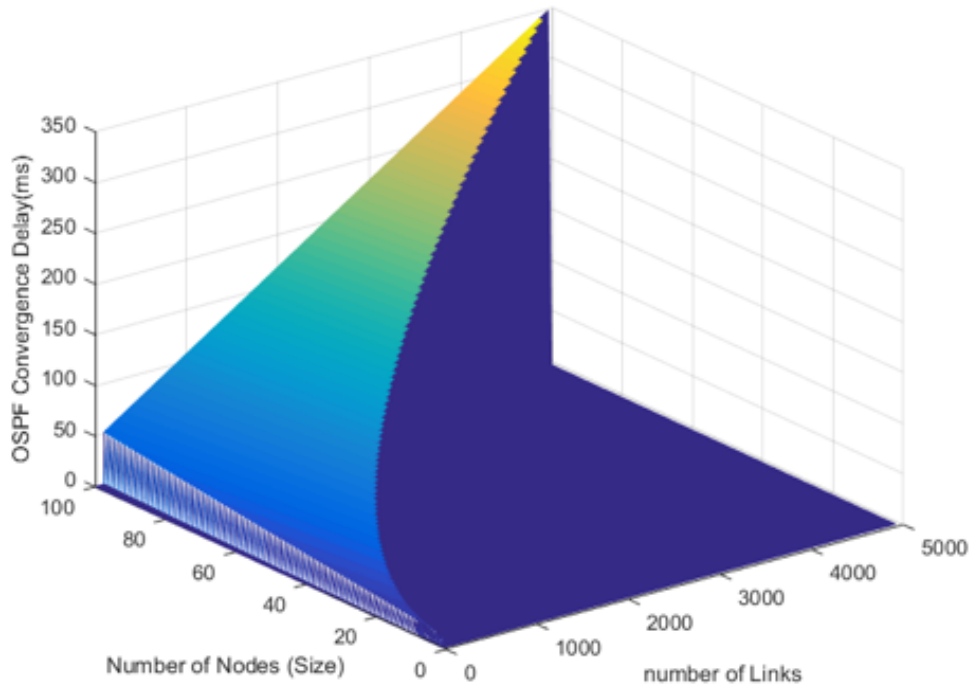


Figure 4.4: OSPF convergence

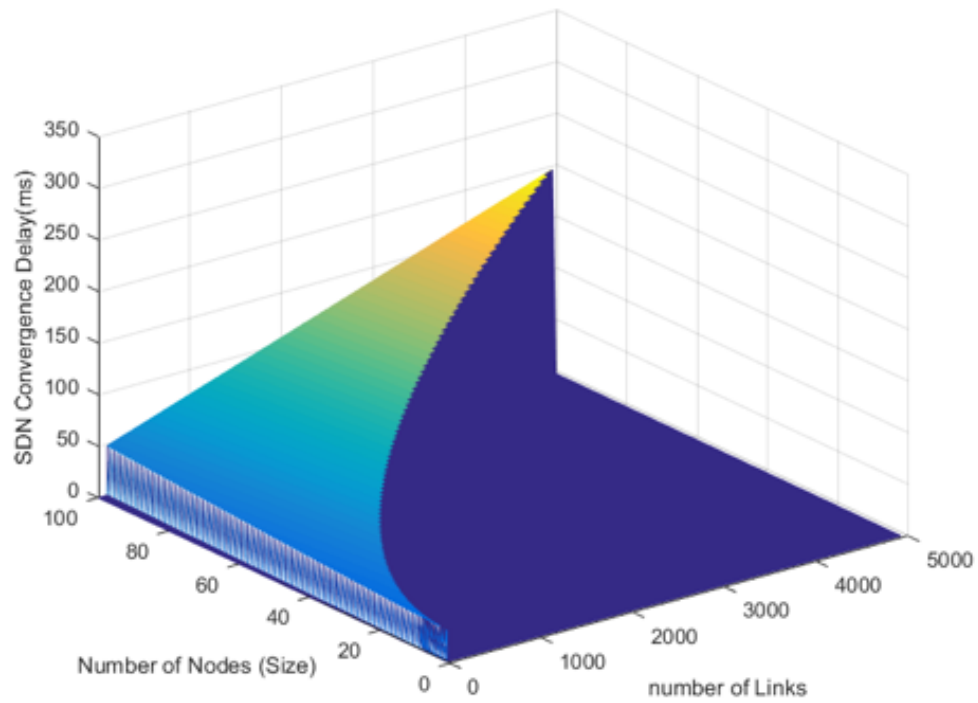


Figure 4.5: SDN convergence

controller with the PortDownReconciliation module enabled. This module deletes switches flows at port down event, which allows us to optimize the controller response. In fact, instead of waiting for the rules to expire on the switches before asking for new forwarding rules, the switches can directly inquire about forwarding assistance from the controller.

Measuring Network Convergence

Convergence was triggered by bringing down a link and measuring the time it takes for the network to be connected again. To test network convergence, we chose the longest path in the network, and assumed that if this path converged, then the whole network has converged. Additionally, there is no straightforward technique to measure path convergence in a network. Many techniques were presented and used in the literature. In [106], convergence is measured by starting an HTTP server on the destination node, and starting an Httping to the server from the source node on the other end of the network. The convergence delay is the time for the Httping to get restored. However, as mentioned by the authors, this technique involves extra delay due to the HTTP TCP connection setup. Another technique for testing OSPF convergence relies on checking if all routers' routing tables are complete and all destinations can be reached [112]. However, the algorithm for checking the status of the routing tables and comparing them will add overhead delay that will affect the measured convergence value. Also, this technique cannot be used for SDN networks. Consequently, we used the method presented in [107]: we start a continuous ping from the source host to the destination node in the network, and start packet capture at the destination. We then bring down a link, and watch traffic interruption and then recovery. To find the convergence delay, we verify the sequence numbers of the received

packets, and calculate the delay based on the missing sequence numbers and the frequency of the pings. The testing script pseudocode is shown in Algorithm 1.

```

Data: Topology, Configuration
Initialization: Start Mininet Network;
if Topology is OSPF-Topologies then
  | Apply-OSPF-Configurations(configuration)
else
  | Apply-SDN-Configuration(configuration)
end
while Host1 ping Host2 do
  | Run collection of ICMP replies
  | Bring Link Down
  | Wait for Ping to operate again
end
Stop network
Parse results

```

Algorithm 1: Testing pseudocode

Experimental Results

We tested network convergence for different link conditions, and different fault locations. We implemented experiment 1 and 2 from section 4.1.3, and fixed the network size and number of links as specified in Table 4.5. Results are shown in Figures 4.6 and 4.7.

- Effect of link delay: as expected with OSPF, we observe that as the link delay increases, the convergence delay increases (see Figure 4.6). Also as expected, SDN convergence delay is not correlated to link delay, i.e. the increase in link delay does not automatically cause longer convergence delays. In fact, in the case of the failure of link 43-50 in Figure 4.6, we observe that the network with 5 ms link delay converges faster than the network with 1 ms delay. Also, when link 23-25 fails, the 1 ms delay and the 5 ms delay networks have comparable convergence delays.

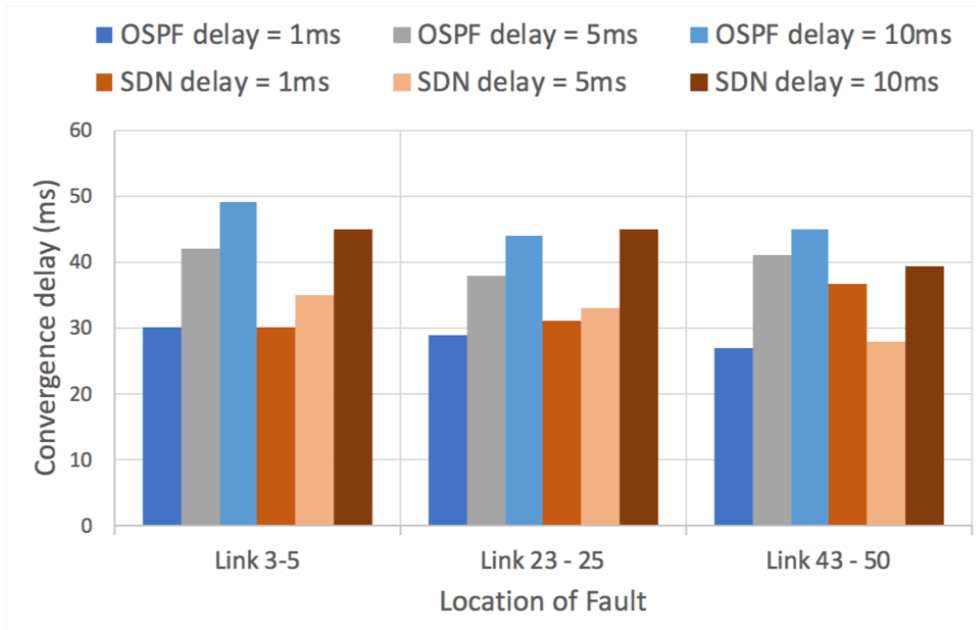


Figure 4.6: Convergence delay in the OSPF network vs. SDN

- Effect of fault location: fault location is expected to affect convergence delay in OSPF because it affects the number of nodes that the advertisements should cross to reach all nodes in the network. However, it is not expected to affect convergence delays in SDN. When comparing the different fault locations in OSPF (see blue and grey bars in Figure 4.6), we notice that for the same link delays (i.e. for bars of same color), the convergence delay is the least when the fault occurs within the network (the link between nodes 23 and 25) as compared to the other fault location. This result is expected as shown theoretically in Figure 4.3.
- Effect of network size: we chose a size of 50 in experimental analysis. In fact, testing OSPF networks of bigger sizes is unfair to the protocol, because an OSPF network of very large size is bad practice and not deployed in real life.

In order to emphasize the delay differences between SDN and OSPF, we re-

peated the experiments with longer link delays of 50, 100, and 200 ms. The results in this case are summarized in Figure 4.7. When the network link delay increases to tens of milliseconds, we are able to observe the convergence advantage of SDN over OSPF. For delays of 100 ms and 200 ms, SDN converges faster than OSPF. For link 43-50 at a link delay of 200 ms, the SDN convergence time is only 60% that of OSPF. Results also show that when the fault occurs inside the OSPF or SDN network (link 23-25), this latter converges faster than when the failure occurs at the extremities (links 3-5 and 43-50). Moreover, Figure 4.7 also shows that if the failure occurs closer to the source (link 3-5), the network, be it OSPF or SDN, converges faster than if the failure had occurred closer to the destination (link 43-50). This is due to the fact that if the failure happens close to destination, the ingress network node at the source will be the last node to be informed of the topology event. Updates will reach source node late because the link delays used are relatively large. Consequently, the ingress node will still forward a packet along the known path until notified of the change. On the other hand, if the failure had occurred at the source, the node would have directly reacted and resolved network convergence, as soon as the first packet is ready to be sent through the network.

Theoretical vs. Experimental Results

Analytical results were obtained using parameters from the experiments. Consequently, we are able to compare experimental and theoretical results. Theoretical and experimental results for OSPF seem to match more closely than those for SDN. In [106], the authors mention that convergence in OSPF networks is related to link delay, whereas convergence in SDN is related to controller performance. However, convergence in SDN does not depend on the controller only. It is also

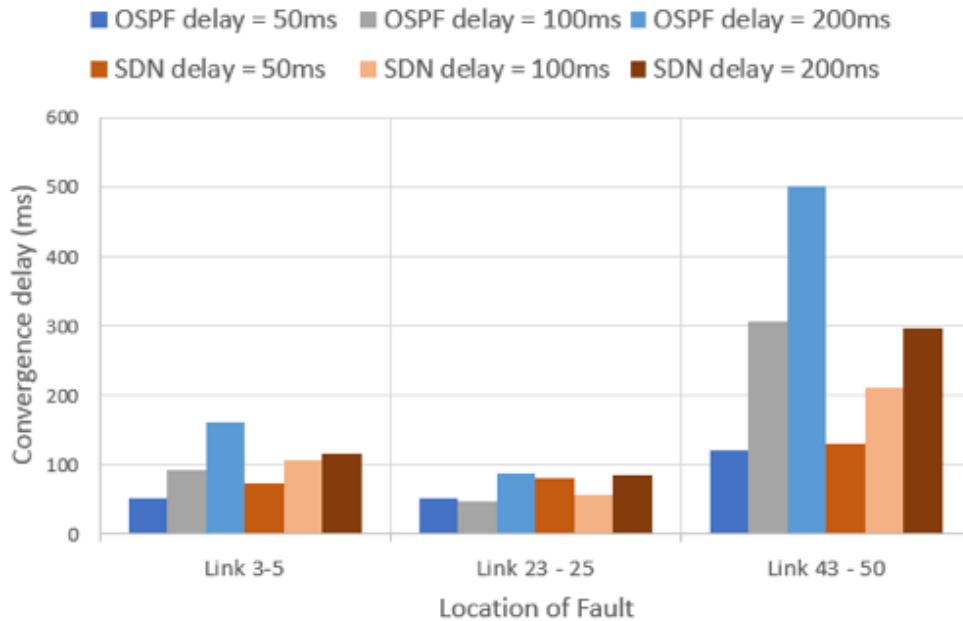


Figure 4.7: Convergence for large network link delays

related to switch performance. In fact, in [121] the authors analyzed the processing latencies of SDN control plane. They consider delays to dispatch rules to switches and the time it takes for the switches to update their flow tables. They also study the interaction between control and data planes, through thorough testing of different vendor switches. They report that the performance of the control plane and the switches is not deterministic, i.e. switches might casually interrupt the processing of control instructions for as long as 200 milliseconds. Also, the authors show a delay between insertion of a rule on a switch and this latter starting to send packets, which means that there is a delay between the control plane and data plane inside the switch itself.

The experimental study verified the validity of the theoretical model, with some irregularities due to the implementation details of the controller, the efficiency of its modules, and the emulation of the switches [121], that are not apparent to operators and users.

4.2 Performance Analysis of SDN vs. OSPF in Diverse Network Environments

In the previous section, we showed the importance of network convergence in networks, since it can limit the damage resulting from network failures and changes. Consequently, a lot of research considered comparing the performance of different routing protocols whether in IP or SDN, to assess their convergence speed and reaction to failures. We modeled OSPF vs. SDN networks in general network deployments and studied their comparative convergence delays for different network conditions. However, the type of network and the architecture choices also affect performance. Consequently we extended the work in the previous section to model network convergence for two network architectures, datacenters and WANs, in the aim of discerning the difference in SDN and IP convergence processes and speeds when the network type and characteristics change.

4.2.1 Extension to the Previous Convergence Model

The model developed in section 4.1 compared SDN and OSPF in random networks under general but ideal settings. The study did not investigate the effect of network type on the performance of the two technologies. However, network type and architecture do have a big impact on the behavior of SDN and OSPF. In fact, SDN was initially conceived for a private Local Area Network (LAN) [122]. It then quickly infiltrated datacenters due to the benefits that it offered to these environments in terms of virtualization and ease of management [7, 123]. In both scenarios, the network is relatively small in terms of diameter, and usually under a single domain authority [23]. Additionally, the work in in section 4.1 gives little care to modeling the SDN control plane, as it assumed that the connection to

the controller is very fast, which is in line with the theoretical definition of SDN. However, in real deployments, connections between switches and controllers are normal network links. Also, the controller/switch pairs do not have to be one hop away from each other, i.e. the controller might not be connected directly to all the switches in the network. Incorporating this aspect within the convergence model provides insights regarding the actual convergence process within SDN.

Datacenters and WANs

We are interested in studying SDN and OSPF in well-defined network types: datacenters (DC) on one hand, and Wide-Area Networks (WANs), on the other. We choose these two very disparate network architectures to emphasize the performance differences of SDN and OSPF deployments. In fact, the two scenarios differ in their architecture setup, latencies, links and connections types, and applications running on them, which will allow us to clearly observe the behavior of the two routing protocols.

Moreover, each technology is favorable to a certain type of networks. For example, SDN is by design datacenter friendly, however, its application to WANs is still in its early stages. In fact, datacenters are usually characterized of being run by a single administrator and with short distances between switches since a datacenter spans a couple of buildings at most, which means hops are only a few meters to tens of meters away from each other [124]. The introduction of SDN to datacenters simplifies their management [125]. It is also proposed to use SDN in datacenters for security solutions to detect and mitigate attacks such as Distributed Denial of Service (DDoS) [126]. Additionally, SDN provides network programmability thanks to virtualization and Application Programming Interfaces (APIs), which makes it easy to run third-party software on the DC

infrastructure [127]. On the other hand, WANs cover large distances, connecting different local area networks. Their deployment is characterized by expensive and complex high-performance core routers and high-speed links to mitigate the effect of big geographical distances [124]. WANs are by nature hard to manage, and they are usually over-provisioned to handle demand spikes. However, SDN offers remote management and virtualization which help solve some problems of the traditional WAN. This avoids overprovisioning by relying on dynamic resource allocation and offers faster reaction to failover [128].

In [129], the authors review the state of the art in SDN deployments in WANs (SD-WAN), with a special interest in the comparison between the challenges of SD-WAN and datacenters. They argue that even though SDN deployment is more frequent in datacenters, the deployment in WANs has recently increased in popularity, especially that WANs are hard to manage. They are also more exposed to failures than datacenters since they are usually overprovisioned. Even though these challenges can be addressed by exploiting the virtualization and efficient network management of SD-WAN, this faces other types of impediments. For example, resiliency to failure is hard to provide with the centralized system especially in large systems such as the WAN. However, many proposals consider SD-WAN to interconnect datacenters, such as Google's 'B4 WAN deployment' [27], which employs SDN to manage the WAN networks that connect different Google datacenters.

In-band and Out-of-band Control

In addition to the network architecture, the deployment of the network technology affects its performance. For example, the choice of in-band or out-of-band control affects the speed of the SDN control plane, which also drove us to con-

sider it in our comparison study. In SD-WAN deployments, the controller can use in-band signaling whereby the control plane utilizes the same links as the data plane, or out-of-band signaling, i.e. separate links to connect the control plane to the switches. In the first case, the design might suffer from performance shortcomings as the control links are not dedicated. Whereas the second scenario incurs additional costs for implementing different channels for the control plane [130]. In datacenters, the control and data planes can share network links, although datacenters usually implement out-of-band SDN control.

Taking into consideration all the above design alternatives, we theoretically model the SDN and OSPF convergence processes, and define the architectures testbeds and scenarios, in the aim of identifying and assessing the behavior of the two routing technologies with respect to all the aforementioned varying network parameters.

4.2.2 Extended Convergence Model

Further in this section, we extend the SDN and OSPF network convergence models of section 4.1 to account for more realistic scenarios and architectural specifications (WANs and datacenters). As such, we introduce new variables, and redefine others.

- The link delay \mathbf{d}_1 , observed on the network links, is remodeled as the scalar-matrix product of the matrix of the distances between two nodes Γ , times the reciprocal of link speed, assumed to be $\frac{2}{3} \times \mathbf{c}$, where c is the speed of light. Generally, the speed of the signals on links in wired networks is approximated to $\frac{2}{3}^{\text{rd}}$ the speed of light, due to the permittivity of the material that the links are made of such as copper-polyethylene or fiber optics,

which carry signals at approximately $0.7c$. Γ is specific to the architecture and type of the network.

$$d_l = \frac{\Gamma}{\frac{2}{3} \times c}$$

- The event propagation delay is modeled by the transmission delay at every link and the link delay over all the links in the network. The effective network link bandwidth is given by the product of the link capacity (NLC) and the congestion factor.

$$d_p = \frac{1}{\text{NLC} \times \text{cf}} + \Sigma d_l$$

- We redefine the controller notification delay (d_{CN}), which is the time for the port status message to reach the controller. In fact, in section 4.1, the links between switch and controller were assumed to be of very high speed. However, this does not apply in practice since the links between the controller and switches are mostly normal network links. Also, they do not have to be one hop away, i.e. the controller might not be connected directly to all the switches in the network. Phemius et al. [131] study the effect of these control links on the network performance. The authors compare the underlying Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) protocols effects on the latency and how the implementation of each protocol affects the buffering process at the level of the switches. Consequently, the model of the channel between switches and the SDN controller was introduced to study the effect of the control channel on performance. Therefore, a propagation delay is added to the model to account for the delay observed on the control channels. In our

Variables	
n	Size of the network
l	Number of links
f	Network depth at fault location
d_d	Detection delay
d_p	Link propagation delay
NLC	Network link capacity
d_{SPF}	SPF calculation time
IPS	Instructions per second
cf	Congestion factor
d_l	Network Link delay matrix
d_{FOU}	Forwarding table update delay
CLF	Controller load factor
linkBW	Bandwidth of links between controller and switches
d_{prop}	Propagation delay on control plane
d_{CN}	Controller notification delay
d_{RF}	Rules dispatching delay
d_{FLU}	Flow table update delay

Table 4.6: Variables and parameters

design, the transmission delay is modeled as the inverse of the bit-rate or bandwidth (linkBW) between controller and switches, which is defined as the product of the link bandwidth and the controller load factor at time t . The propagation delay is modeled as the distance matrix divided by speed, which is also assumed to be $\frac{2}{3} \times c$. Consequently:

$$d_{CN} = \frac{1}{\text{LinkBW} \times \text{CLF}} + d_{prop}$$

Where $d_{prop} = \frac{\chi}{\frac{2}{3} \times c}$ and χ is the distance vector between the controller and the SDN switches, which is specific to the simulated network type and architecture.

All variables are summarized in Table 4.6.

Given these additions, we define the convergence process of OSPF and SDN

as follow:

OSPF Convergence Time

$$\text{ConvOSPF} = d_d + d_p + d_{\text{SPF}} \times 2 + d_{\text{FOU}} \quad (4.1)$$

$$\text{ConvOSPF} = d_d + \frac{l}{(\text{NLC} \times \text{cf})} + \Sigma d_l + \frac{\#\text{inst}}{\text{IPS} \times 2} + d_{\text{FOU}} \quad (4.2)$$

SDN Convergence Time

$$\text{ConvSDN} = d_d + d_{\text{CN}} + d_{\text{SPF}} + d_{\text{RF}} + d_{\text{FLU}} \quad (4.3)$$

Retaining the assumption that rules are broadcasted all at once:

$$\text{ConvSDN} = d_d + \left(\frac{1}{\text{LinkBW} \times \text{CLF}} + d_{\text{prop}} \right) + \frac{\#\text{inst}}{\text{IPS}} + \left(\frac{1}{\mu} + n \times \max(d_{\text{CN}}) \right) + d_{\text{FLU}} \quad (4.4)$$

Furthermore, we define additional equations to model convergence delays when the controller is in-band:

$$\text{ConvSDN}_{\text{IB}} = d_d + d_{\text{CN}} + d_{\text{SPF}} + d_{\text{RP}} + d_{\text{FLU}} \quad (4.5)$$

where d_{RP} is the delay for rule propagation in the network and is defined by the sum of the transmission of the rules on the control network, their propagation to the switch which is directly connected to the controller, \hat{n} , and the rules' propagation within the network, as follows:

$$d_{\text{RP}} = \left(\frac{n}{\text{LinkBW} \times \text{CLF}} + d_{\text{prop}}(\hat{n}) \right) + d_p \quad (4.6)$$

Consequently,

$$\begin{aligned} \text{ConvSDN}_{\text{IB}} = & d_d + \left(\frac{1}{\text{LinkBW} \times \text{CLF}} + d_{\text{prop}} \right) + \frac{\#\text{inst}}{\text{IPS}} \\ & + \left(\frac{n}{\text{LinkBW} \times \text{CLF}} + d_{\text{prop}}(\hat{n}) \right) + d_p + d_{\text{FLU}} \quad (4.7) \end{aligned}$$

4.2.3 Implementation and Testing

We model realistic architectures of both WANs and datacenters to test our convergence model. The topologies are shown in Figures 4.8 and 4.9, respectively. Table 4.7 specifies the real distances between the WAN sites shown in Figure 4.9. We investigate both in-band and out-of-band SDN controllers for both architectures. In datacenters, the location of the controller is not critical since the distances between the switches are very small. Consequently, they will not have a big impact on traffic propagation in the network. In this case, the controller is assumed to be connected to switch s1, as shown in Figure 4.8. For the WAN, these distances become considerable. This is why for the WAN experiment, we examine two scenarios: In the first scenario, the controller C1 is located at the edge of the network (connected to switch s10 as shown in Figure 4.9). In the second scenario, the controller C2 is located in the middle of the network (connected to switch s7 as shown in Figure 4.9) to check the effect of controller placement especially for in-band control.

Simulation Setup

This model accounts for the delays incurred on the control plane links, i.e. the links between controller and switches. In fact, in practice, these links are usually the same type of links as network links, rather than fast links. The connection

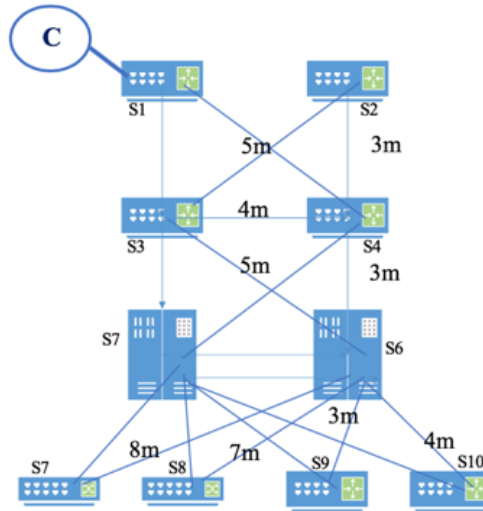


Figure 4.8: Datacenter layered architecture and distances

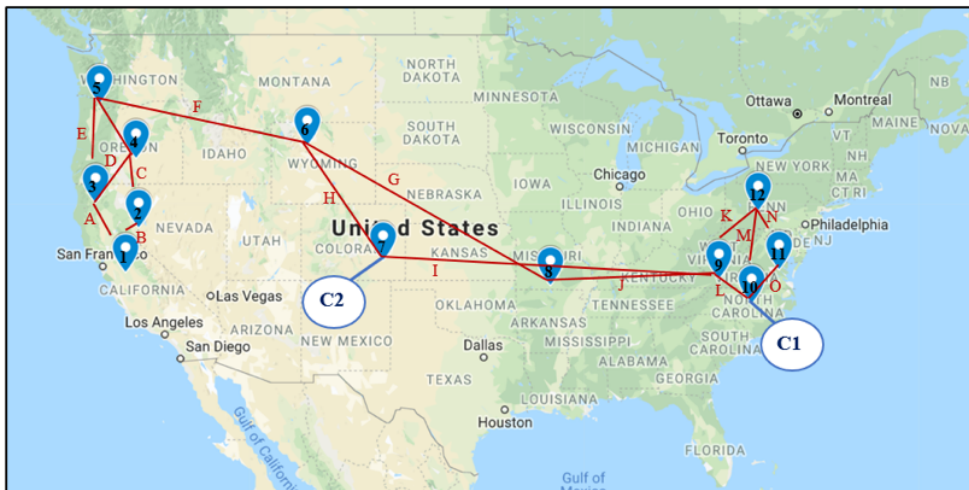


Figure 4.9: Wan architecture (background image taken from maps.google.com)

Link	Distance
A	444 km
B	261 km
C	406 km
D	325 km
E	527 km
F	1,107 km
G	1,587 km
H	790 km
I	2,000 km
J	1,000 km
K	463 km
L	244 km
M	553 km
N	368 km
O	260 km

Table 4.7: Distances between the WAN sites shown in Figure 4.9

Experiments
1- Effect of fault location on convergence
2- Effect of network congestion
3- Effect on controller load
4- Effect of controller capacity
5- Effect on controller/switch link speed
6- Controller capacity vs. network congestion
7- Effect of in-band vs out-of-band controllers

Table 4.8: Experiments

can also be logical, i.e. links do not have to be direct connections between the controller and switches; they may be hops apart. This extended model reflects the behavior of such logical links by introducing a delay matrix for the control plane channels. The model is tested on the two network architectures (shown in Figures 4.8 and 4.9) under the scenarios reported in Table 4.8. The distance matrices for network links (Γ) and the distance vectors between the controller and SDN switches (X) are defined as follows:

$$\Gamma_{\text{Datacenter}} = \begin{bmatrix} 0 & 0 & 3 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 5 & 0 & 4 & 3 & 5 & 0 & 0 & 0 & 0 \\ 5 & 3 & 4 & 0 & 5 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 5 & 0 & 4 & 4 & 3 & 7 & 8 \\ 0 & 0 & 5 & 3 & 4 & 0 & 8 & 7 & 3 & 4 \\ 0 & 0 & 0 & 0 & 4 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 4 & 0 & 0 & 0 & 0 \end{bmatrix} \text{ in meters.}$$

$$\Gamma_{\text{WAN}} = \begin{bmatrix} 0 & B & A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & C & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 0 & D & E & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & C & D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & E & 0 & 0 & F & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & F & 0 & H & G & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & H & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & G & 0 & 0 & J & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I & J & 0 & L & 0 & K \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L & 0 & O & M \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & O & 0 & N \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & K & M & N & 0 \end{bmatrix}$$

where quantities A to O are specified in Table 4.7 in km.

$$X_{\text{Datacenter}} = [5 \ 5 \ 7 \ 7 \ 10 \ 10 \ 13 \ 12 \ 12 \ 13] \text{ in meters}$$

$$X_{\text{WAN-C1}} = [3.5 \ 3.5 \ 3.7 \ 3.7 \ 4 \ 1.5 \ 2.3 \ 1.2 \ 0.3 \ 0.05 \ 0.3 \ 0.6] \text{ in km}$$

$$X_{\text{WAN-C2}} = [1.1 \ 1.2 \ 1.3 \ 1.3 \ 1.5 \ 0.8 \ 0.05 \ 0.85 \ 2.02 \ 2.2 \ 2.4 \ 2.6] \text{ in km}$$

Results and Analysis

The analysis of results follows 2 axes: comparing the performance of OSPF to that of SDN under different network conditions on one hand, and identifying the effect of the type of network on the performance of the two protocols, on the other. In the graphs below, we plot the convergence delay versus different network parameters as per Table 4.8. In some of the experiments, we represent the axes by the inverse of the variable in question (marked by a -1 on the axis unit), to emphasize the variations of the curves in the graphs, especially when the intersection between the OSPF and the SDN curves is not easily detectable. We first study the effect of fault location in both networks in experiment 1. We can see from Figures 4.10 and 4.11 that the location of the fault does not affect the convergence delays in both datacenters and WANs. Although the result is expected for SDN, the behavior of OSPF was not in accordance with the previous results, which showed increasing convergence delay for OSPF when the fault moves to the network extremities (Figure 4.3). This is due to the fact that the predefined architectures are small in size (10 nodes for datacenter network and 12 for the WAN network), unlike the previous general network architecture which contained 50 nodes.

However, the second experiment (Figures 4.12 and 4.13) shows that network

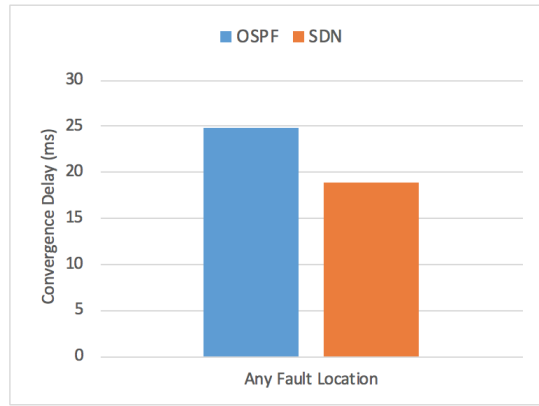


Figure 4.10: Fault location effect on convergence in datacenter

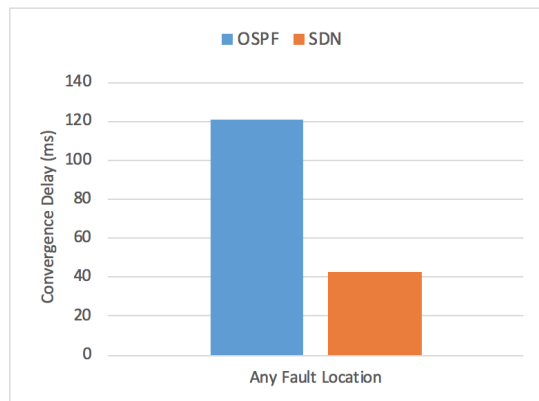


Figure 4.11: Fault location effect on convergence in WAN

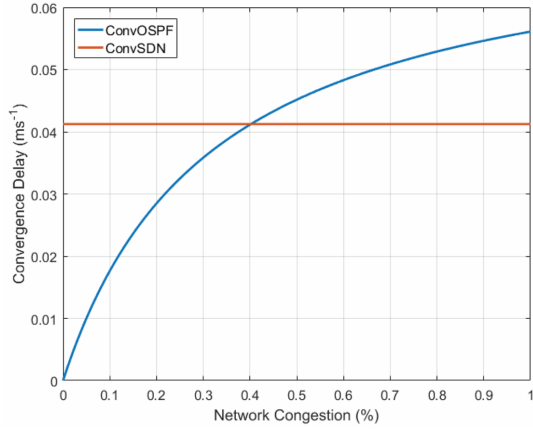


Figure 4.12: Network congestion effect on convergence in datacenter

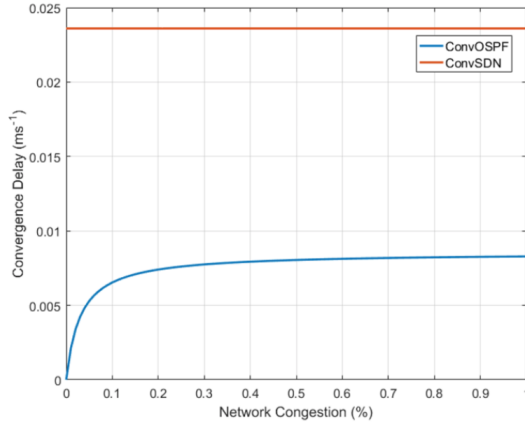


Figure 4.13: Network congestion effect on convergence in WAN

congestion affects OSPF but not SDN, which explains the horizontal red lines in the Figures. This is expected because rippling the advertisements in OSPF is highly affected by the congestion on the network links. Alternatively, SDN uses a separate control plane, which makes the control operations independent from the network data links. Additionally, Figure 4.13 shows that SDN always outperforms OSPF in WANs, whereas in datacenters, OSPF outperforms SDN when the congestion factor is larger than 0.4, which corresponds to traffic less than 60% of the normal traffic condition (Figure 4.12).

In Experiments 3 to 5, we are controlling variables related to the SDN con-

troller in both SDN and OSPF networks. Obviously, these variables will not affect convergence of OSPF because the network does not include a controller. However, the main idea is to observe the behavior of SDN when these variables change and discern when SDN performs better than OSPF for these changes. For example, when we vary controller capacity, we are looking to see if/how controller capacity can make SDN perform better than OSPF.

The Controller Load Factor (CLF), which is tested in Experiment 3, does not affect convergence in both types of networks (Figures 4.14 and 4.15). In fact, both the datacenter and WAN networks are small, and thus the controller doesn't have a large number of switches to service. This is why CLF has very little effect on the performance of the SDN network. However, when we compare SDN to OSPF, we observe that SDN converges faster in both the datacenter and the WAN. The difference in convergence speed is larger for the WAN than it is for datacenters. This is mainly because the distances separating nodes in the WAN incur link delays that hinder OSPF convergence speed. In Experiment 4, the controller capacity, unlike the controller load, does have a big impact on the convergence delay in the SDN network. In fact, we observe in Figures 4.16 and 4.17 that the delay is exponentially reduced as the controller capacity increases; the more capacity a controller has, the faster it can service switches, which reflects positively on the convergence speed. When comparing the datacenter to the WAN, we notice that SDN outperforms OSPF faster in the WAN (at about 5% of the maximum capacity, compared to 60% for the datacenter). Alternatively, when testing for the bandwidth (BW) between the SDN controller and the switches for normal network operation in Experiment 5, we deduce that the BW doesn't affect the convergence performance (straight lines for increasing BW capacity in Figures 4.18 and 4.19) as soon as it starts increasing. This is expected in the

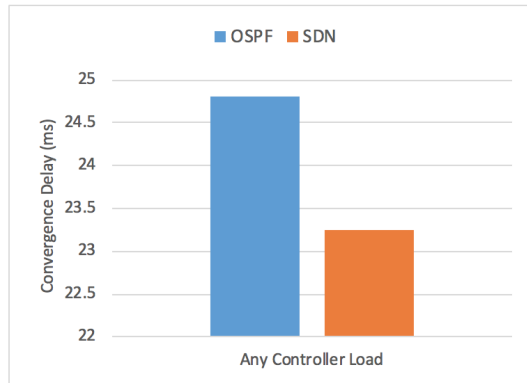


Figure 4.14: Controller load effect on convergence in datacenter

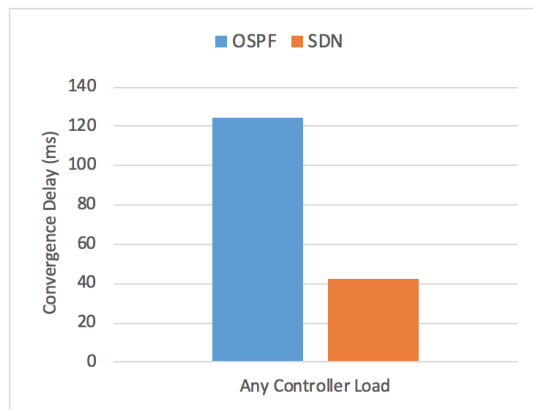


Figure 4.15: Controller load effect on convergence in WAN

architectures we defined because they include a small number of SDN switches (10 in the datacenter and 12 in the WAN). Consequently, regardless of the volume of the control data, the BW does not constitute a bottleneck to network operation, except when the controller is not reachable (which corresponds to 0 BW).

Experiment 6 consists of studying the effect of network congestion as controller capacity increases, to identify any correlation between the two variables. When comparing the behavior of OSPF to that of SDN in both Figures 4.20 and 4.21, we observe that OSPF convergence is correlated to network congestion, because for increasing values of network congestion, the OSPF convergence delay increases.

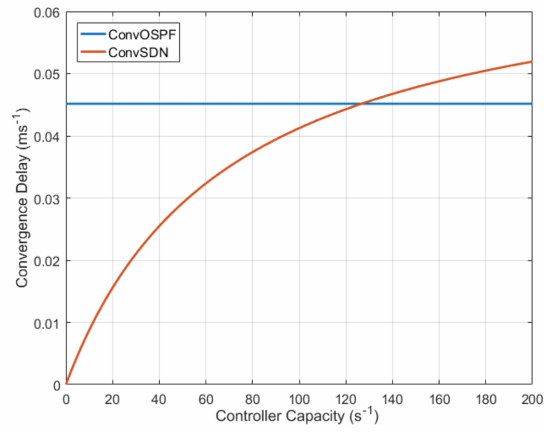


Figure 4.16: Controller capacity effect on convergence in datacenter

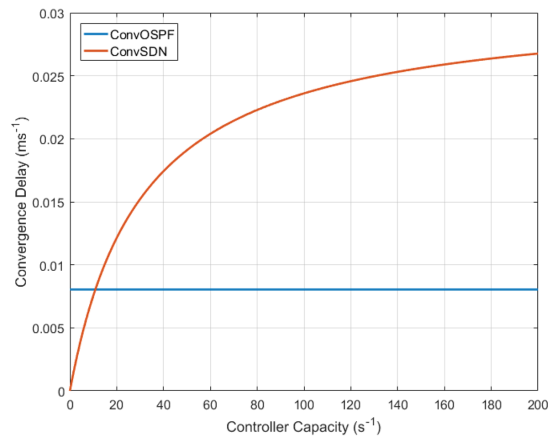


Figure 4.17: Controller capacity effect on convergence in WAN

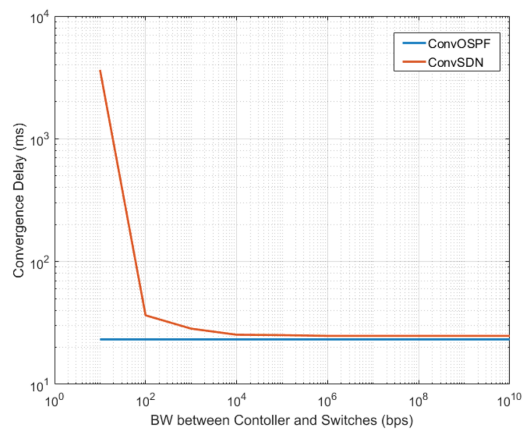


Figure 4.18: Control link BW effect on convergence in datacenter in log-log scale

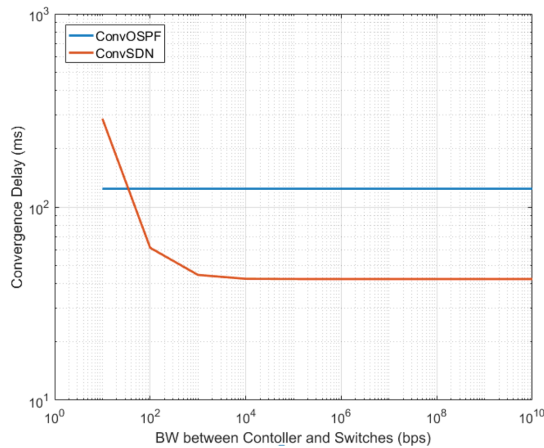


Figure 4.19: Control link BW effect on convergence in datacenter in log-log scale

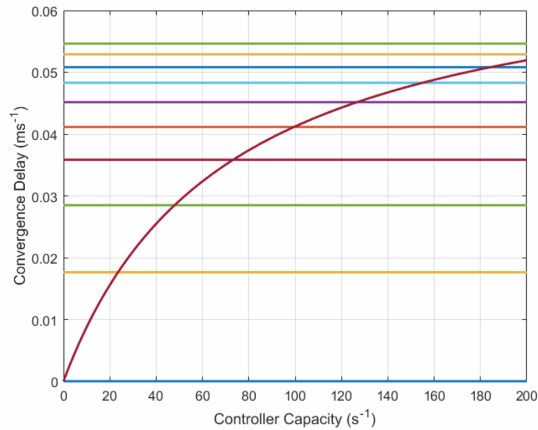


Figure 4.20: Controller capacity effect for different network congestion values in the datacenter

This delay is independent of the controller capacity (as expected, as the OSPF network doesn't include an SDN controller) since it is constant for all values of the x-axis. However, we notice that even though the behavior of SDN is dependent on the controller capacity, it is not affected by network congestion. In fact, we observe on the graphs the superposition of SDN convergence delay curves for different values of network congestion, while the convergence delay decreases as the controller capacity increases.

Experiment 7 studies the effect of in-band controllers. As previously men-

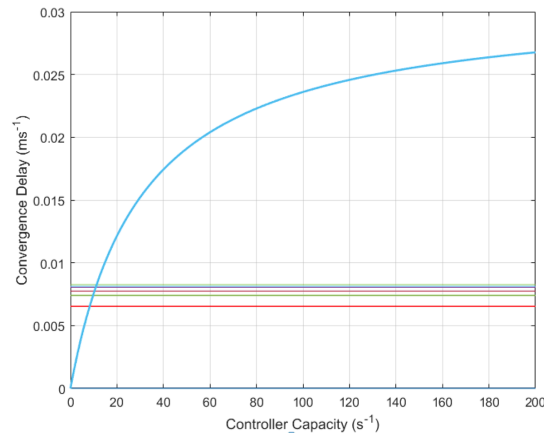


Figure 4.21: Control capacity effect for different network congestion values in WAN

tioned, for the datacenter network we are modeling, the in-band controller is physically connected to one switch in the network, which in turn relays the control information to its neighboring switches in a mechanism similar to message rippling in OSPF. Consequently, the controller capacity does not have an effect on the convergence speed, which explains the straight yellow line in Figure 4.22. However, we observe that the in-band controller performs worse than OSPF. This is because in addition to the message rippling that is common to SDN and OSPF, there exist additional delays related to sending the rules from the controller to the connected switch(es). We tested two controller locations for the WAN network: network extremity and center. Results are shown in Figure 4.23. Again, the convergence of the networks with in-band controllers is independent of the controller capacity. However, their location affects their convergence speed. In fact, OSPF outperforms SDN with in-band controller when this latter is on the edge of the network. However, OSPF is greatly outperformed by SDN with an in-band controller located in the middle of the network. In this scenario, the distance needed to deliver the message to all switches is half of the distance of the whole network, which explains why it is faster than when the controller is

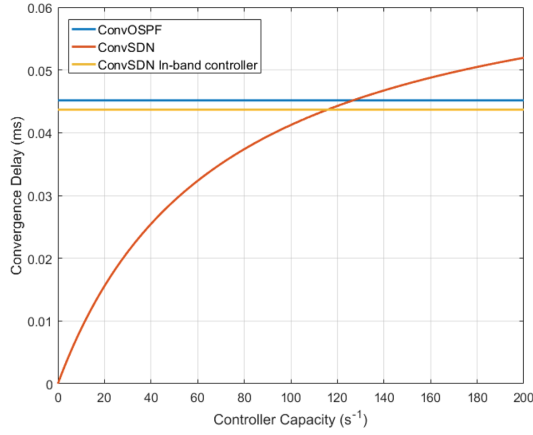


Figure 4.22: Effect of in-band and out-of-band controller on convergence speed in the datacenter

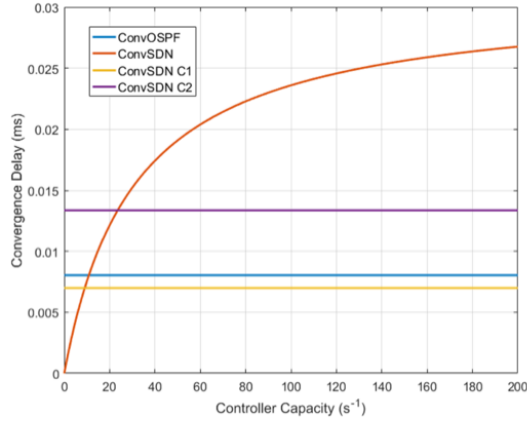


Figure 4.23: Effect of in-band and out-of-band controller on convergence speed in the WAN

connected to an edge switch.

Experimental Results

In addition to the analytical simulation, the architectures defined in Figures 4.8 and 4.9 are implemented in Mininet [120] with the real distances defined in $\Gamma_{\text{Datacenter}}$ and Γ_{WAN} . We use the same method as the one used in the previous section to test convergence speed. It consists in running a continuous ping from source to destination, bringing a link down on the flow path in the network, and

collecting missing sequence numbers at the destination to compute the convergence delay. In this experiment, the distances between the network nodes are defined as per Figures 4.8 and 4.9, and network variables are set to their default values in Mininet. Given that the distances in the datacenter are very small, the delay on the links is extremely small. Consequently, as the time to convergence is faster than our machine resolution, this latter was not fast enough to capture the convergence delay in datacenter; we were not able to collect experimental results for this architecture. Alternatively, Figure 4.24 shows results comparing OSPF and SDN convergence for the predefined WAN architecture. We notice that SDN converges faster than OSPF in the WAN for all the trials performed. These results are in line with our analytical findings of section 4.2.3. In fact, when observing Figures 4.11, 4.13, and 4.15, we see that in the WAN SDN delivers faster performance all the time for all values of the varied parameters. When the varied parameter is the capacity of the SDN control plane, Figures 4.17, 4.19, 4.21, and 4.23 also show that SDN delivers better performance than OSPF does except when this capacity is small, as expected. In fact, when the SDN control planes becomes a bottleneck, SDN will certainly perform badly.

We also study the effect of the delay of the control plane on the convergence delay (Figure 4.25). As expected for OSPF, results show that the convergence delay is approximately constant, reaching an average value of 2.4 ms. Alternatively, for SDN, as the delay to controller increases, the convergence delay increases, until a given delay (272 ms) for which SDN becomes slower to converge than OSPF. In fact, the slower the control links, the longer it takes for control packets to be exchanged between the switches and the controller; consequently, the longer it takes for the network to converge.



Figure 4.24: Convergence delays of SDN and OSPF in the WAN

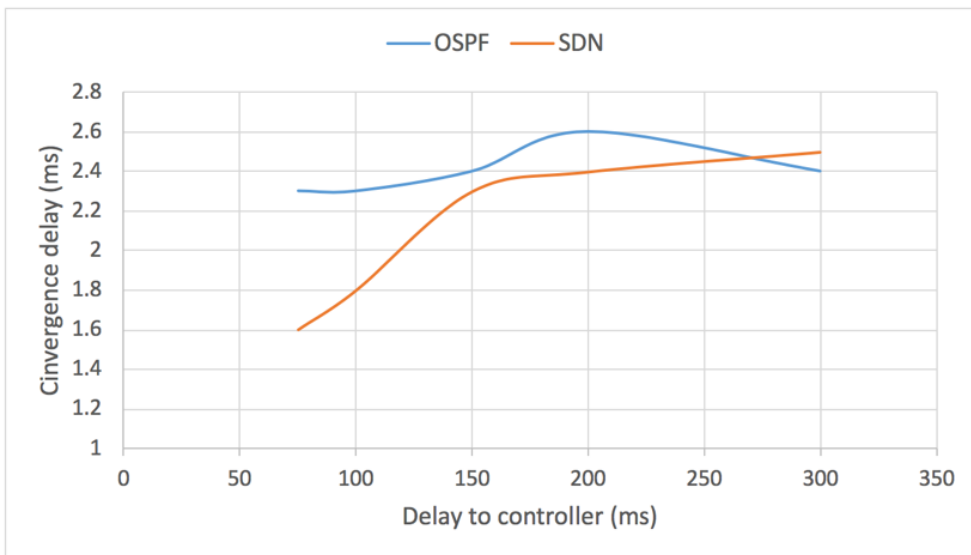


Figure 4.25: Effect of the control plane delay on convergence in SDWAN

4.2.4 Discussion and Analysis

Improvement over preliminary work

Given that the preliminary results were obtained from a general network architecture, we investigate the effect of the network and architecture type on the performance of the two protocols. This justifies our choice of two disparate networks, the WAN and the datacenter. We extend the previous model to be applied on the two network types, and test the protocols' convergence performance accordingly. This incurs many enhancements over the preliminary design. First of all, the exact topology and the distances are fed to the model, instead of assigning the number of nodes only like for the old model. This gives better representation of reality and better precision, since it allows us to locate the fault instead of relying on the general modeling of the depth of a network ($O(\log n)$) as in the previous model. Second, the simulation/implementation is done using real network parameters, eg. real distances, delays, etc. In addition to architectural enhancements, the model also has better accuracy since it includes the representation of the control channel. In fact, in real SDN deployments, the control plane links are regular network links that suffer from transmission and propagation delays. In the preliminary work, the control links were assumed to be very high-speed links with zero delay. This work also extends the previous experimental setting: in Matlab, the testing scenarios include additional tests on parameters that weren't considered before, such as the effect of network congestion, of the controller load, of the control links speed, and the effect of in-band versus out-of-band control in SDN. As for the simulation on Mininet, a test for the effect of the control links delay on the WAN and datacenter is conducted in addition to the comparative test for convergence delay.

Key Observations

The work in this chapter concerns the routing performance of two control plane technologies: centralized control plane with SDN and distributed control plane with IP networks deploying the OSPF routing protocol. We first start by studying the performance of the protocols in a general network architecture. The model is first tested on Matlab, and then simulated on Mininet for a random architecture consisting of 50 network elements. In the first set of experiments, the two protocols outperform each other with changing network conditions. For example, network link delay affects convergence delay of OSPF more than that of SDN. Consequently, for low link delay, OSPF converges faster than SDN. However, as the delay increases, OSPF becomes slower to converge. These results are expected, since the convergence process in OSPF relies on the network links to advertise changes whereas in SDN the control plane is used to update the routing information in the switches. Other parameters like fault location have the same effect as the link delay; OSPF convergence suffers as the fault occurs more towards the extremity of the network, whereas SDN convergence is not affected by fault location. As for the experimental testing, the network parameters are set in Mininet to match the parameters in Tables 4.2 to 4.4. Comparing analytical to experimental results allowed us to validate the general model. Essentially, experimental results show that convergence of OSPF is affected by the network link delays whereas SDN convergence is not, which matches the finding of the analytical testing. Also, when it comes to fault location, experimental results show that OSPF is more affected by the location of the fault than SDN, which is also in line with the analytical finding.

Similarly, we test the enhanced model analytically and experimentally. For the analytical testing, we run scenarios in Matlab to observe the effect of different

network parameters on the convergence speed. When testing for fault location, the behavior was different from that of the general network. In fact, in the extended work, fault location didn't have an effect on the convergence of OSPF, unlike what was previously observed. As justified earlier, this is due to the fact that the architectures of the WAN and the datacenter include less nodes than the general network architecture. As for network congestion, we observe different behavior in datacenter and WAN: in the datacenter, OSPF has better convergence up to almost 60% congestion, whereas SDN always outperforms OSPF in the WAN. The other parameters (i.e. controller load, controller capacity, and bandwidth between controller and switches) are related to SDN networks, but the experiments allow to find the break-even point at which one technology starts performing better than the other. The controller load didn't affect convergence speed because the simulated networks are already small in size, however high controller capacity and high BW allowed SDN to converge faster than an OSPF network running under the same network conditions. The SDN control type and placement problem is also evoked: we compare the effect of in-band and out-of-band control on the performance of SDN with respect to performance of OSPF. We conclude that a network with in-band SDN control behaves like an OSPF network, with extra overhead due to the delay it takes for the controller to flush the rules to the connected switch. The location of the in-band controller also affects SDN performance: the more the controller is placed towards the center of the network, the better the performance of SDN. Results from these experiments aren't always in accordance with the results from the general model, which justifies the extension of the previous model, and validates our hypothesis that in addition to network parameters and circumstances, the performance does depend on the network type and architecture.

In this section, we extend the OSPF and SDN convergence models to account for control plane delays, and compare the two for important network types, namely datacenters and WANs, to study the effect of the network type and parameters on the performance of the protocols. Results show that OSPF and SDN behave differently in datacenters and WANs, and that SDN and OSPF have varying convergence speeds within the same network type. In other words, in datacenters like in WANs, the two protocols outperform each other for different values of network variables. Although one can prefer a technology over the other, it all comes down to the network design and characteristics which actually affect the performance of the protocols and deliver better convergence speeds.

Chapter 5

Model of Adaptive Control in Hybrid Systems

The evolution of the control plane in communication networks was driven by the needs of each time period. For example, PSTN was introduced following telephone exchange to account for scalability. Also, ATM was created to address the need of carrying voice and data. Alternatively, control in nervous systems and political systems is affected by the status quo and the nature of the events taking place. For example, in the nervous systems, critical events that requires fast response (eg. reflexes) are handled with the distributed control, whereas in political systems, critical events that require shrewd decisions (eg. decision to go into war) are handled with central control. These guidelines can be generalized and applied to any system that exhibits both type of control. Referring to section 2.5 and Figure 2.5, we rely on the defined allocation of control tasks to determine the control rules for our proposed optimization systems below.

5.1 Offline Recommender System

The first proposed design for a technology recommendation system was an offline decision making system based on fuzzy logic. The motivation behind the work was related to the fact that the network operator is left with many choices when deploying or updating a network. However, with the new options of network forms, the decision becomes less straightforward. Many factors come into play aside from performance such as financial issues, which further complicates the situation. This first design consisted in a fuzzy decision system that embodies network deployment best practices according to the advantages gained with each technology (legacy and SDN), and provides technology and financial recommendations that help operators decide on what type of network is best to deploy given their network and service requirements, in the light of its effect on the company's CAPEX and OPEX. The tool recommends the type of network to opt for based on a set of parameters that the operator advises.

5.1.1 Fuzzy Decision System Design

A fuzzy system is a good candidate for the network recommendation tool since it deals with many uncertain and vaguely defined variables [132]. For example, when talking about network size, there are no well-defined boundaries for the size categories. In other words, we are very well certain that a network with 2 nodes is considered as a small network; however, a network with 10 nodes can be considered either small or medium. The system then responds with defined actions such as the choice of technology to use. Therefore, the system is modeled as a Fuzzy Inferences Systems. In fact, all input variables that come into play in this case are fuzzy; they do not show clear cut limits between their ranges.

This applies to all of the inputs of the decision making block such as network size, utilization, frequency of updates, percentile of services deployed, or initial technology deployment. Network size (NS) is considered as an input to the Fuzzy System because SDN and IP networks perform differently with the network size. Consequently, the choice of technology depends on this variable. The size can be Small, Medium or Large depending on the number of nodes in the network. The smallest network can be composed of one node; therefore, one is the low limit of our variable. The network can be as large as needed: the upper limit on the variable is infinity. The ranges are defined as follows:

- Small (S): 1 – 15
- Medium (M): 10 – 50
- Large (L): 40 – Infinity (200)

The network utilization (NUT) or traffic load is another parameter we consider. In fact, when utilization is high, the central control plane can be overloaded due to the increased level activity of the switches [133]. NUT is denoted as a percentage of the maximum network capacity; the ranges are defined as:

- Very high (VH): 85 – 100 An extremely high network capacity target is a bad network design. Consequently, the utilization threshold is usually set below this range.
- High (H): 50 - 90
- Average (A): 35 - 70
- Low (L): 0 - 40

Network updates (NUP) can be very infrequent or extremely frequent. NUP directly influence the choice of technology because SDN provides an easy programmable interface for updating network nodes, and removes the burden of individually configuring routers [134]. The Frequency of these updates is classified in an interval of 0 to 12 months, the minimum being once every 12 months (once a year) and the maximum is once every 0 month or daily updates. The ranges of updates are given as follows:

- Low (L): once every [6 – 12] months
- Average (A): once every [3 – 7] months
- High (H): once every [1 – 4] months
- Very High (VH): once every [0 – 2] months

Initial Technology Deployment (ITD) refers to the state of the existing network. This parameter is considered as an input to the Decision System because the initial deployment influences the state of the final network. ITD can be SDN, Heterogeneous, or Traditional ranging from:

- Traditional (T): 0 % are SDN nodes
- Heterogeneous (H):]0-100[% are SDN nodes
- SDN: 100% are SDN nodes

The Ease of Management (EM) relates to the number and types of services that are required in the network. We quantify it with percentages, from hardest (0%) to easiest (100%). A network is hard to manage if it supports many technologies such as MPLS, VxLAN, or Middle-boxes, which are inherently hard to manage. On the other hand, the network is easily manageable if it doesn't

implement any extra service (other than routing). This parameter also affects performance because the harder the network is to manage, the better it is to move towards SDN, given the advantages in network management and control that this latter provides [134]. The ranges are defined as follows:

- Low (L): 0 – 35
- Average (A): 30 – 70
- High (H): 65 – 100

In our design, the output is the technology to choose for the new network upgrade. The network technology choice (NT) can be:

- SDN: 100 – 90% SDN switches
- HighHeterogeneous (HH): 95 – 45% SDN Switches
- LowHeterogeneous (LH): 55 – 5% SDN Switches
- Traditional (T): 10 – 0% SDN Switches

5.1.2 Defining the Inference Rules

The inferences rules are defined as follows:

1. If $ITD == SDN \ \&\& \ EM == L \ \&\& \ NUT == (L||A)$
 $\rightarrow NT = HH$
2. If $ITD == SDN \ \&\& \ EM == L \ \&\& \ NUT == (H||EH)$
 $\rightarrow NT = LH$
3. If $ITD == SDN \ \&\& \ EM == H$
 $\rightarrow NT = SDN$

4. If $ITD == SDN \ \&\& \ EM == A \ \&\& \ NUP == (H||VH)$
 $\rightarrow NT = SDN$
5. If $ITD == SDN \ \&\& \ EM == A \ \&\& \ NUP == A \ \&\& \ NS == L$
 $\rightarrow NT = HH$
6. If $ITD == SDN \ \&\& \ EM == A \ \&\& \ NUP == A \ \&\& \ NS == M$
 $\rightarrow NT = SDN$
7. If $ITD == SDN \ \&\& \ EM == A \ \&\& \ NUP == A \ \&\& \ NS == S$
 $\rightarrow NT = SDN$
8. If $ITD == H \ \&\& \ EM == L$
 $\rightarrow NT = LH$
9. If $ITD == H \ \&\& \ EM == H$
 $\rightarrow NT = SDN$
10. If $ITD == H \ \&\& \ EM == A \ \&\& \ NUT == (H||VH) \ \&\& \ NS == L$
 $\rightarrow NT = T$
11. If $ITD == H \ \&\& \ EM == A \ \&\& \ NUT == (H||VH) \ \&\& \ NS == M$
 $\rightarrow NT = LH$
12. If $ITD == H \ \&\& \ EM == A \ \&\& \ NUT == VH \ \&\& \ NS == S$
 $\rightarrow NT = HH$
13. If $ITD == H \ \&\& \ EM == A \ \&\& \ NUT == H \ \&\& \ NS == S$
 $\rightarrow NT = SDN$
14. If $ITD == H \ \&\& \ EM == A \ \&\& \ NUT == A \ \&\& \ NS == M$
 $\rightarrow NT = HH$

15. If $ITD == H \ \&\& \ EM == A \ \&\& \ NUT == A \ \&\& \ NS == L$
 $\rightarrow NT = LH$
16. If $ITD == H \ \&\& \ EM == A \ \&\& \ NUT == A \ \&\& \ NS == S$
 $\rightarrow NT = HH$
17. If $ITD == H \ \&\& \ EM == A \ \&\& \ NUT == L \ \&\& \ NS == S$
 $\rightarrow NT = SDN$
18. If $ITD == H \ \&\& \ EM == A \ \&\& \ NUT == L \ \&\& \ NS == (L||M)$
 $\rightarrow NT = HH$
19. If $ITD == T \ \&\& \ EM == (L||A)$
 $\rightarrow NT = SDN$
20. If $ITD == T \ \&\& \ EM == H \ \&\& \ NUT == (H||VH)$
 $\rightarrow NT = T$
21. If $ITD == T \ \&\& \ EM == H \ \&\& \ NUT == (A||L)$
 $\rightarrow NT = SDN$

5.1.3 Financial Implications of the System Decisions

Technological decisions are directly coupled with financial implications. In fact, the decision variables that drive the technology choice also affect CAPEX and OPEX, which are in turn influenced by this choice.

CAPEX and OPEX Applied on Decision Variables

CAPEX mainly concerns buying infrastructure and network software [135], but it also includes cost of training and acquiring knowledge. Consequently, Network size and Utilization are correlated to CAPEX since any change in any of

these variables induces an investment in infrastructure (routers, switches, links and so on) which directly affects CAPEX. On the other hand, OPEX includes all expenses related to the operation of the network, such as installation, service delivery and maintenance [135]. In this case, Utilization, frequency of Network Updates and Ease of Management explicitly influence OPEX; a high Utilization or a high Frequency of Updates require more Network Maintenance and consequently more expenses, whereas Ease of Management is related to Service Provisioning and the Easier the network to operate, the less incurred expenses.

As the network variables influence costs and expenditures of companies, the decisions that are based on these variables also influence CAPEX and OPEX. In our design, the inferences rules thrive to reduce OPEX and keep the increase in CAPEX as low as possible in order to stay in line with companies' financial preferences. In fact, companies seek to reduce their OPEX and invest in CAPEX because reducing their expenses allows them to stay competitive in the market (price-wise) and grow market shares. It also increases their benefits which makes them more attractive to investors [136]. The recommendation of the Fuzzy System improves OPEX because the technical decisions are deliberately crafted to reduce downtime and bottlenecks and increase the efficiency and the ease of handling of the network. On the other hand, CAPEX is basically affected by the purchase of new equipment and technologies, and training for employees. SDN can contribute in reducing the increase in Capex because the technology is cheaper than legacy boxes [137]. Furthermore, training is only required when operators are introducing a new technology to the network, because if the initial network is SDN or hybrid, then the personnel are already familiar with SDN. We are assuming that any network employee already knows how to operate traditional networks, and training for that matter is not necessary at this point.

Cost, Management Overhead, and Performance Gain Evaluation

A low ease of management means many services deployed. Some services, such as middle boxes, are more efficient than their SDN equivalent. Therefore, if the initial network is SDN, choosing a heterogeneous network where most of the nodes are SDN but the ones that are particularly related to those services are traditional nodes transforms the network into a more efficient one. Investment in CAPEX takes place because of the purchase of new legacy network boxes, but OPEX is kept low because the efficiency of the network is increased (Rule 1). In addition to the above analysis, when utilization is high, it might overload a centrally controlled network. A distributed approach is more appropriate in this case, which explains the choice of Heterogeneous where SDN nodes do not outnumber traditional ones. CAPEX is also affected because of the cost of new regular network boxes. However, OPEX is kept low because operating costs are reduced due to and increased efficiency and a reduced risk of failure (Rule 2). On the other hand, when the ease of management is high, it means that very few services are implemented; the recommendation is to extend the network with SDN. In this case CAPEX is only affected if the operator chooses to buy extra SDN nodes, but OPEX is also kept low because managing and running the network are easier thanks to SDN (Rule 3).

When the ease of management is average in an SDN network, then SDN could be adopted if the update frequency is high or very high. In this case, the increased complexity in dealing with many services is mitigated by the fact that SDN greatly simplifies the update procedure. In this case OPEX is kept low because managing and running the network are easier thanks to SDN. However, when the updates are not very frequent, the benefit of having SDN for updates exceeds the shortcomings of the average numbers of services only when the network size

is small or medium. When the network is large, it is better to opt for a High Heterogeneous network. This decision affects CAPEX because middle boxes are being purchased. Again, OPEX is protected because an eventual overload (that incurs cost of repair and loss) on the controller is avoided (Rules 5, 6, 7). When the initial network is hybrid, CAPEX is not affected, or only slightly affected if the company decides to buy extra hardware, because the network already contains both types of appliances. Training is not needed because employees are already exposed to both types of networks. As for OPEX, the effect of Update Frequency, Ease of Management and Size of Network is the same as above (Rules 8 to 18). However, if the initial network is traditional, CAPEX is influenced not only due to the purchase of new SDN equipment, but also because of the investment in SDN training for personnel. The variation is kept minimal because SDN technology is cheap. OPEX is kept low as per the above analysis as well (Rules 19, 20 and 21).

Financial Recommendation

The System provides recommendation for technology deployment complemented with the financial implication of the decision. However, the degree of variation in OPEX and CAPEX is specific to each network, and a case by case study is required in order to deduce exact figures and numbers.

Some companies criticize the increase in CAPEX on the ground that funds are not always available for investment. In this case, a company can mitigate the effect of high OPEX by leasing the equipment for a certain number of years, which transforms CAPEX into OPEX.

5.1.4 Implementation and Case Studies

The Fuzzy System is implemented using MATLAB. The variables and the inferences rules defined above were integrated into the fuzzy model. Since our system features 5 inputs, visualization of the results is done by fixing 3 of them, and studying the variations of the output decision with respect to the other 2 variables. The Mamdani inference method is used to model the fuzzy inference system. Fuzzification of the input variables is performed using triangular membership functions, because they provide an average degree of membership for the variables [138]. Additionally, defuzzification at the output is implemented using the centroid method.

Case Studies

Figure 5.1 shows how the decision varies with size and utilization when frequency of updates is low to average (once every 6 months), an average number of services are implemented (50%), and the initial network is heterogeneous (50% SDN nodes).

As seen in the figure, for the above 3 conditions, when the network size is large and the utilization is high, opt for an addition of traditional nodes (output of 0). In fact, when the network size is large and utilization is high, having also a good number of services deployed will overload the controller. This latter is receiving a large flow of packet-in (due to high utilization), and communicating with a large number of switches (large network), but also has to run a good number of services.

The controller becomes a point of failure and the risk of network shortage becomes high. Also the delay becomes higher because the single resource is being drained. In this case, it is better to opt for traditional nodes even because this will

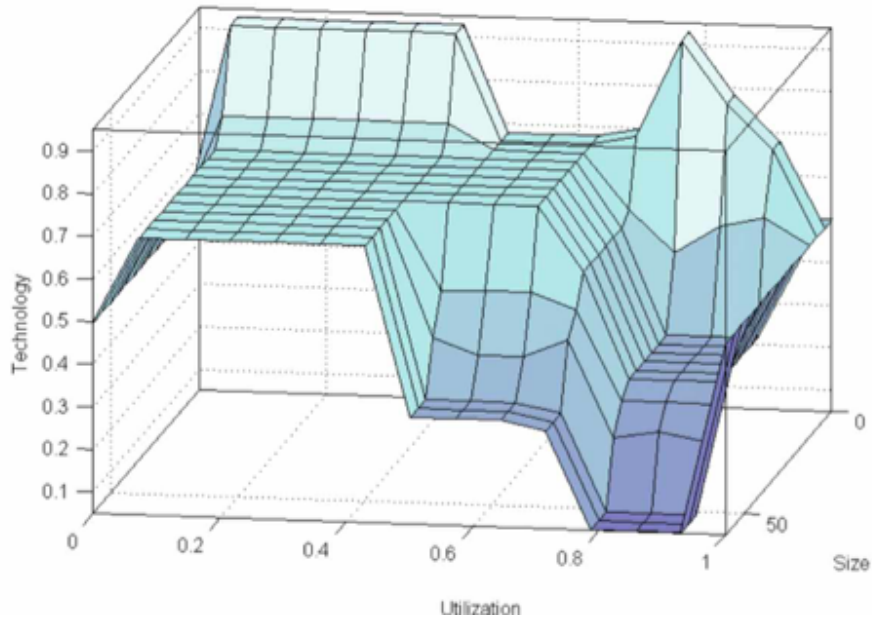


Figure 5.1: Case study 1

slightly increase network management at the sake of better network performance and network availability. Consequently, OPEX is protected and CAPEX is only increased by the price of the new legacy routers.

The second case study shows more contestable situations; these do not correspond to any of the specified inference rules, and they illustrate the benefit of using a fuzzy system. Figure 5.2 illustrates an example.

In this scenario (case 2), frequency of updates is low to average (once every 6 months), the initial network is heterogeneous (50% SDN nodes), but the ease of management is low to average (33%). In this case, rules cannot specify what output to obtain because the ranges of the input variable are within the gray zone, for example we are not sure if 33% is low or average ease of management. Consequently, the fuzzy system representation for those sets of input is required in order come up with a decision.

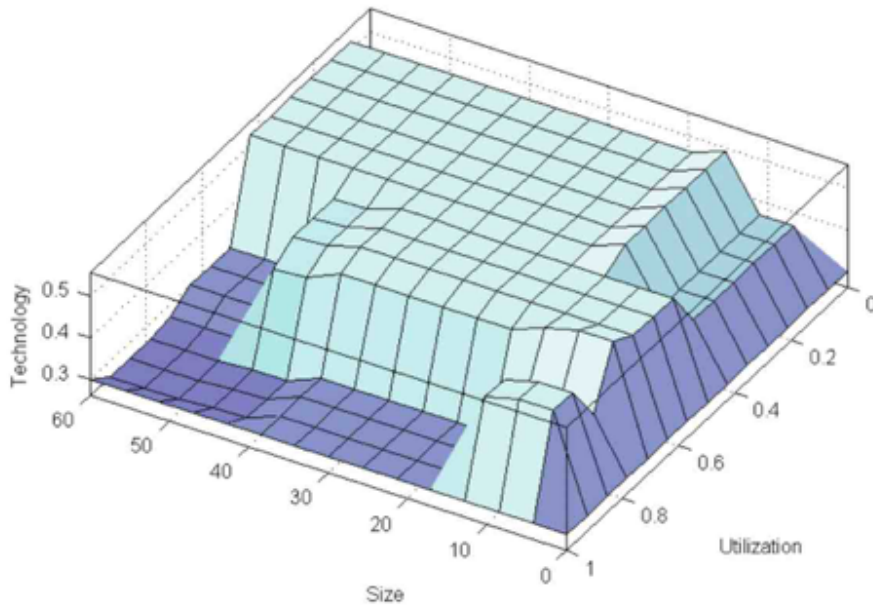


Figure 5.2: Case study 2

Progressive Scenario

Given an initial Traditional network, where the Ease of Management is Average and the Update Frequency is Average, the Fuzzy System output the recommendation shown in Figure 5.3.

If the Utilization is low (20%) and the Network Size is Medium (20), then the recommendation is to opt for a High Heterogeneous network with high ratio of SDN switches versus traditional ones. In fact, even though number of services is average, updates are moderately frequent and the network size is average, the combination of all three in traditional networks incurs high management and control costs.

Consequently, it is better to start introducing SDN in this network, especially that the utilization is low which means that the controller will not be drained with high packet-in/packet-out activity. SDN can be used to implement the

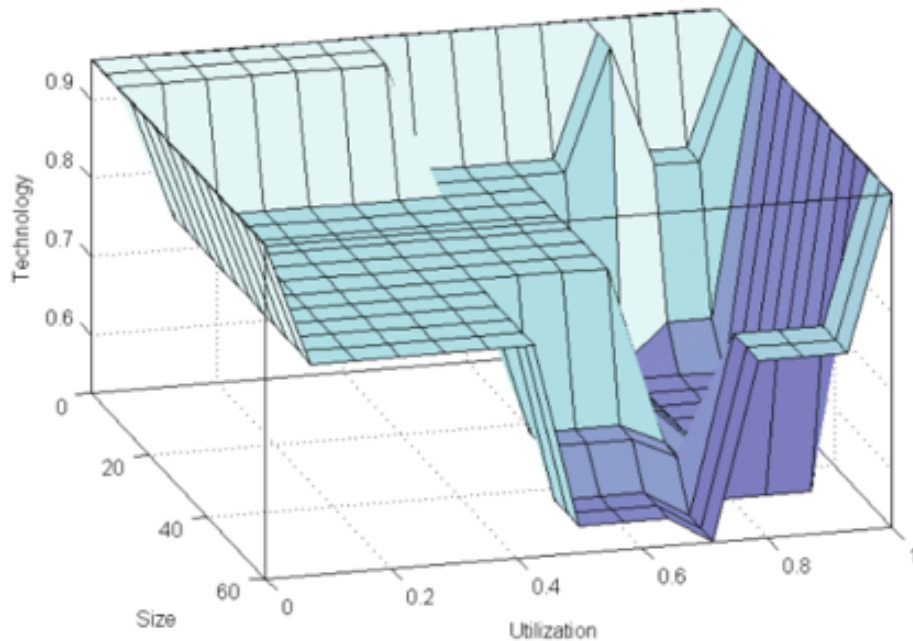


Figure 5.3: Progressive scenario: initial state

services that are easier to manage with SDN like MPLS, and help in providing a more automated approach to network update thanks to the resulting partially-centralized control. Cost increase and CAPEX are kept minimal because the price of SDN switches is lower than traditional routers, even if there was a slight investment in employees training. OPEX is also protected because SDN provides more effective network and service management which reduces operating costs.

On the next network extension, the current network is a hybrid network, where the Ease of Management is Average and the Update Frequency is Average (assuming that these variables are kept constant). The Fuzzy System outputs the recommendation shown in Figure 5.4.

Again, if the utilization is low (20%) and the size is medium (30, because we added SDN switches on the previous round), the recommendation is also High Heterogeneous. This scenario affects cost, management overhead and performance similarly to the previous one: Management is more efficient; CAPEX and

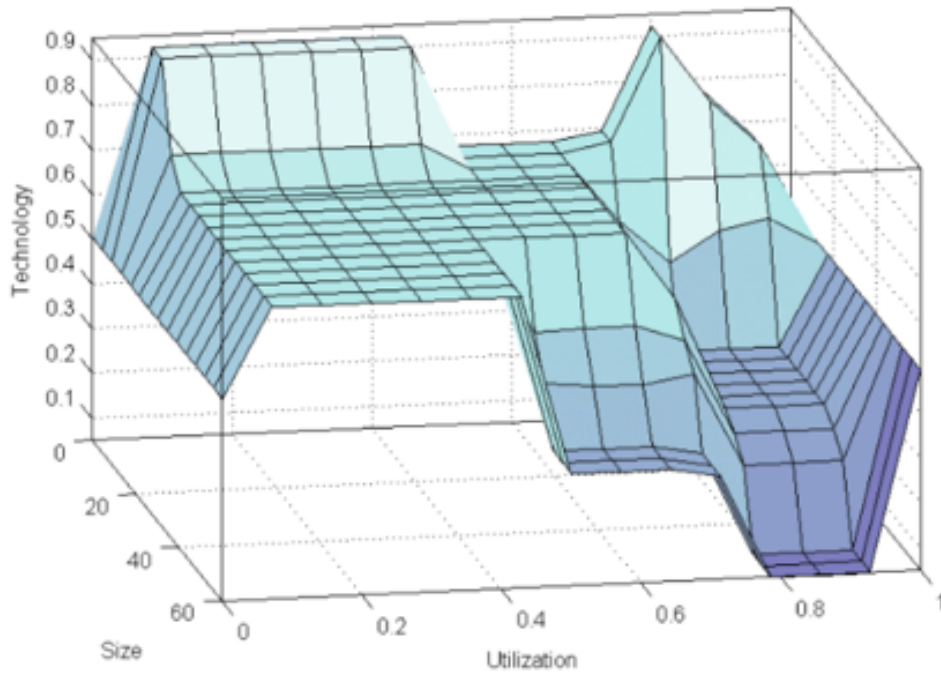


Figure 5.4: Progressive scenario: round 2

OPEX are kept minimal. Employee training is not required because it already took place at the previous network update (previous scenario). We notice from Figure 4 that for that given network with the same Ease of Management and Update Frequency, the recommendation is always High Hybrid. So we reached a steady state. This steady state is maintained as long as the Ease of Management, the Update Frequency, and the Utilization are kept constant. If the update frequency increases for example, then the recommendation of the Fuzzy System is as in Figure 5.5.

This option is feasible because on the previous 2 updates the network was moving towards high SDN which makes it easy for the operator to move to full SDN. This is expected because SDN is recommended when the update frequency is high. Consequently, as the network shifts towards SDN, the increase in CAPEX is minimal because it concerns only the price of new SDN switches which are

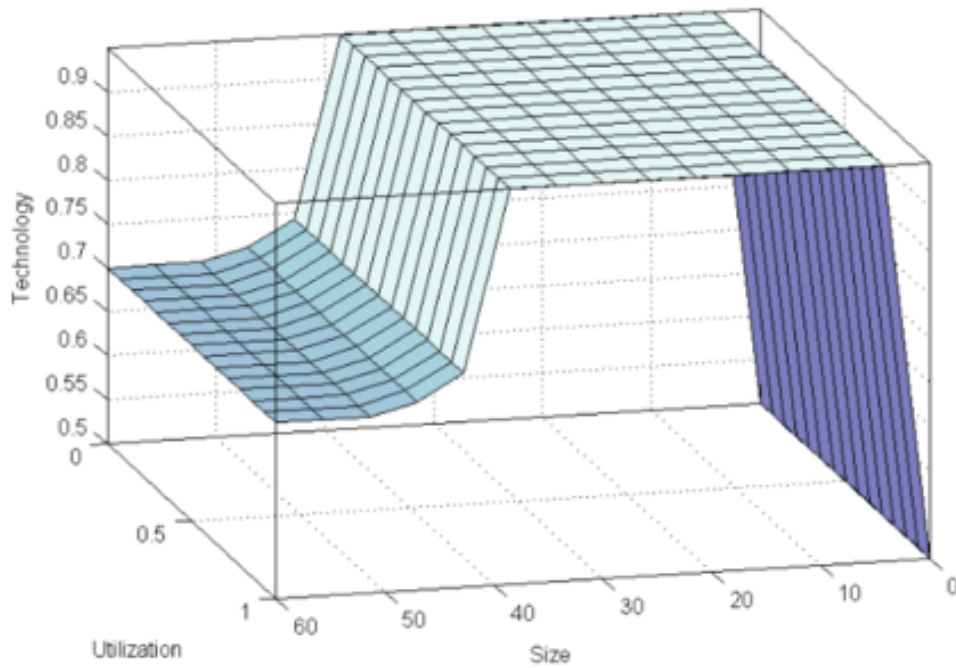


Figure 5.5: Progressive scenario: change in update frequency

cheap. Training is not required because the personnel is already trained and has been working with SDN switches for two updates now. OPEX is also protected because a high update frequency in a medium sized network would have required more engineering and configuration man-hour in traditional or hybrid networks than in SDN, which would have drastically increased OPEX and management overhead.

The decision system developed in this subsection is an offline decision making system that is ran once, before network deployment, to decide on the type of technology to opt for. The subsequent decision making systems are online system that run every period and update the control state of the switches according to the results obtained.

5.2 Modeling the General Optimization System

We define a general system that embeds distributed control and centralized control, corresponding to distributed Agents and one Central Authority (CA) respectively. The Agents compose a network of Agents, which we also refer to as the System Environment, and all Agents can communicate with the CA. In this system, both the CA and the Agents can play the role of the control plane; in case of the CA, the control plane is centralized, whereas in the case where Agents do the control, the control plane is distributed.

Based on the allocation rules, we generalize the division of control tasks that are identified in Figure 2.5: We designed an algorithm in which the system decides if control decisions will be centralized or distributed, based on an optimization problem that aims to maximize system robustness. System *Robustness* as defined in this work is a property relating to three basic metrics: *Availability*, *Reliability*, and *Scalability*. Consequently, the control rules that we follow maximize one or more of these metrics. In other words, agents adaptively switch their control plane from centralized to distributed given network conditions. Control rules are inspired from the interaction between central and distributed control in the different systems previously discussed.

The decision-making system can be in turn computed centrally at the CA level, or in a distributive manner at the Agents level. It can also be in itself hybrid, by combining a sub-optimization (DO) that is run in a distributive manner on the Agents, on top of which runs a global optimization (GO) on the CA, in a manner similar to a bi-level Optimization. The location of each module of the Optimization System (OS) with respect to the System Environment is shown in Figure 5.6, and the high-level optimization model is shown in Figure 5.7.

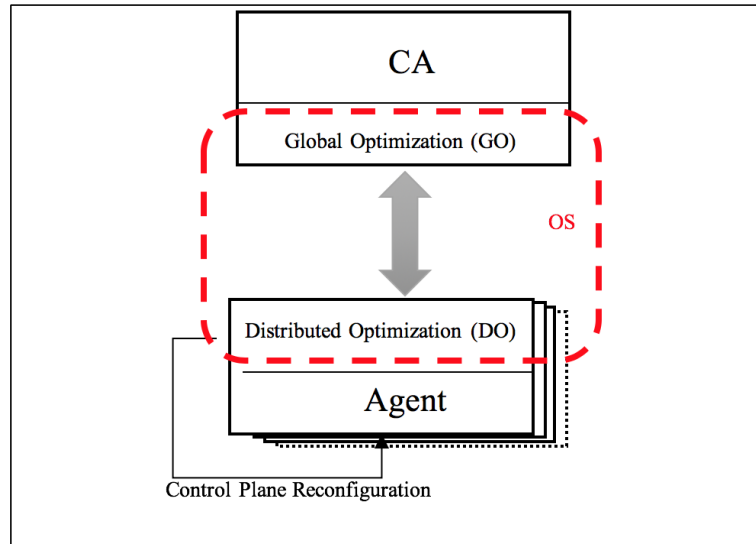


Figure 5.6: OS emplacement within system environment

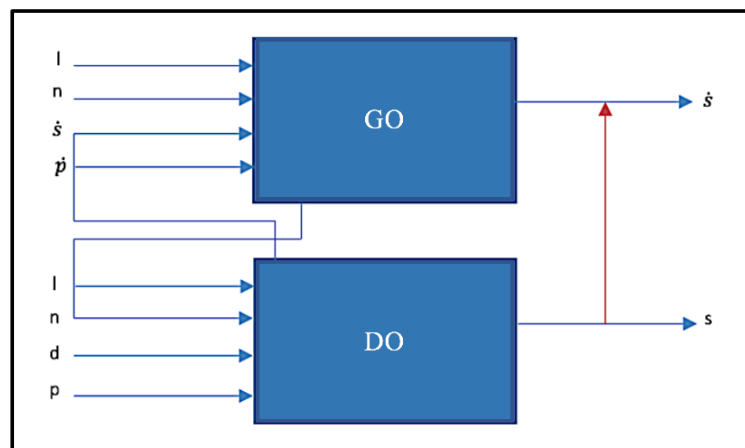


Figure 5.7: Optimization general model - GO: global optimization, DO: distributed optimization

In this optimization, the DO algorithm is invoked at every T_o period on each Agent. Alternatively, the Agent recomputes its optimization algorithm at any event occurrence advertised by its neighbor or by the CA. To avoid excessive computations, we define a Hold-off Timer T_{HoldOff} , which specifies the delay that the Agent must wait before recomputing the algorithm upon an event notification. T_{HoldOff} is defined by design; it should be smaller than the period T_o . Results from the DO of all Agents are passed on to the GO, which computes the percentage of distributedness of the system, taking into account the results of the Agents.

The variable of the optimization problem is the control state of the agent, and the supervised variables (system parameters) are defined as follows:

1. The CA response time to any communication received from an agent. It represents the CA availability, reachability or responsiveness.
2. The stability of the environment; ie. rate of changes in the network.
3. The uncertainty in the environment, or the number of unknown events that occur in the network.
4. The size of the environment.

The control rules are defined as follows:

5.2.1 HCA Cost

The health of the CA is evaluated by its responsiveness. The associated cost is measured by the delay it takes the CA to reply to the Agents.

*Control rule 1: When the CA delay increases, instruct the Agent to rely less on the CA and go more distributed, to ensure **Availability**. Similar to the situation*

in political science, when the central authority fails, power is transferred to state governments [11].

5.2.2 SH Cost

The health of the system (SH) is evaluated by its *stability*. Control rule 2: When the rate of changes in the network increases, instruct the Agents to rely on the CA.

*If the topology of the network keeps changing, the CA provides a comprehensive view of the system that makes it react better to these changes and ensure higher system **Reliability**.*

5.2.3 Uncertainty Cost

The Uncertainty cost (UC) is evaluated by the characteristics of the events: known, unknown, or abnormal.

*Control rule 3: When the rate of unknown information increases, the Agent should rely more on the central control since this latter has better knowledge, providing thus improved network **Reliability**.*

5.2.4 SS Cost

Large systems perform better when intelligence is distributed.

*Control rule 4: When the size of the system increases, the Agent is better off being more distributed. Distributed algorithms provide more **Scalability** in this case.*

As previously defined, Robustness is achieved by improved Availability, Reliability, and Scalability. These metrics are in turn defined as the combination of

the 4 control rules defined above.

5.3 Application to Telecom Networks:

5.3.1 Adaptive Hybrid System (AHS): Overview and Design

Centralization and decentralization of control exists in today's network. While IP still rely on distributed control and routing algorithms, SDN emerged as the centralized guru. Consequently, the model developed in the previous section can be applied to networks when they feature both types of controls. In fact, SDN removes the burden of distributed route calculation at the node level, which provides network administrators more control over their systems [7]. However, Link state routing protocols are more scalable and respond better to changes in the routing path; they provide reliability and fault tolerance [3]. Additionally, the SDN controller is a single point of failure, and redundancy should be enforced in an SDN network to avoid network shortage in case of a controller failure [8]. Essentially, each of the two types of control has its own advantages and disadvantages, and hybrid networks promise to merge as the best of both.

We apply the above general model to networks to leverage the advantages of SDN (centralized control plane) and IP (distributed control plane). In this scenario, we refer to our system, which includes the optimization system and the controlled hybrid network, as the Adaptive Hybrid System (AHS). The CA is the SDN controller, and the Agents are hybrid switches (HS). These switches support both Centralized and Distributed routing protocols. They can be either controlled by the SDN controller in a centralized manner, or they can operate as regular

routers and make their own routing decisions in a distributed fashion. We assume that the network of HS is an overlay solution, and that all switches are virtually connected, forming peer adjacency. Also, the HS communicate information about their own network.

The Hybrid Network (HN) Design

In our design, the HN is composed of homogeneous switches that natively support both centralized and distributed routing protocols, the Hybrid Switches (HS), and of the SDN controller. The SDN controller is assumed to be the central authority that is omnipresent and has full knowledge of all aspects of the network. The HS's can be centrally controlled by the SDN controller or they can act autonomously, in which case they become capable of making their own routing decisions. This is due to the fact that the HS's support both the IP protocol stack and the OpenFlow protocol, which enables them to understand both types of signaling.

The forwarding table on Hybrid Switches is the compilation of forwarding rules learnt from the distributed and the central control planes to ensure consistency among both planes. In fact, in our design, the controller is in charge of maintaining the consistency of both control planes (centralized control plane (CCP) and distributed control plane (DCP)) in the network. The controller has a full view of all network elements including the HS's with distributed control plane, and can control any HS at any time no matter what state it is in. Actually, the HS's are instructed to send LLDP packets to the controller, which can be used by the controller to construct the full network map. The controller adds a rule on all HS's with CCP to broadcast (ripple) OSPF hello messages, to ensure that OSPF advertisements are being received by all HS's. When needed, the controller injects fabricated packets in the network via the HS's. The controller controls

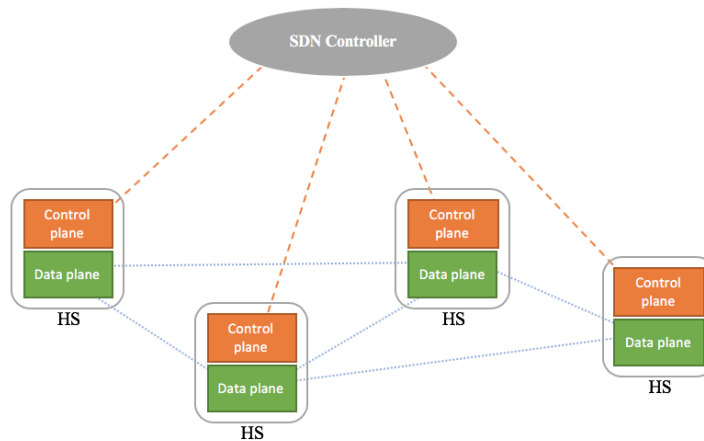


Figure 5.8: The hybrid network (HN)

how OSPF computes the routing tables of an HS with DCP, by instructing the neighboring HS's with CCP to send fabricated routing messages in the network, in a technique similar to [18]. This ensures that both types of control planes are always up to date.

When an HS switches from the DCP state to the CCP state, it sends its routing table to the controller. This latter uses it to update its network map and fabricate the routing messages to be sent to the network. In the opposite case, when the HS switches from the CCP state to the DCP state, the controller sends to the HS all the forwarding information that it needs to operate in the network. Alternatively, when the controller goes down, all of the HS will revert to the DCP state. The network will behave as if it is bootstrapping. Figure 5.8 shows the HN, composed of an SDN controller, and the HS's.

The Optimization System Design

The Optimization System is also hybrid; it is composed of a Global Optimizer (GO) running on the SDN controller, and Distributed Optimization (DO) instances running on each HS. Consequently, there are different design alternatives

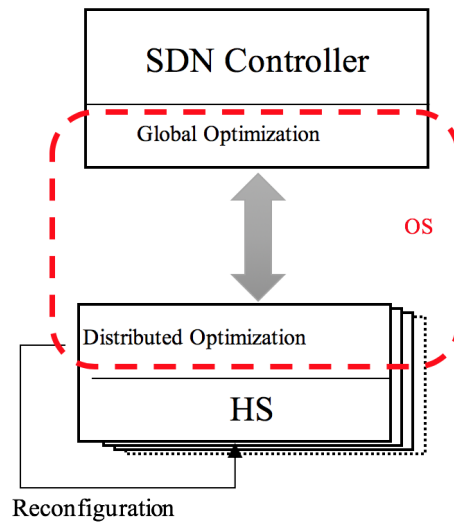


Figure 5.9: OS functional block diagram

to the Optimisation system. The decision to go with CCP or DCP can emanate from the controller when this latter invokes the GO, and instructs the HS's to use CCP or DCP given results of the optimization. Alternatively, each HS can run the DO to individually decide if it wants to use CCP or DCP given network conditions. Local network parameters are monitored on each HS which performs local optimizations, whereas global measurements that are only available at the controller's level (such as network size) are monitored at the SDN controller level. Both local measurements and global measurements are exchanged between the controller and the HS when needed. Figure 5.9 shows the functional block diagram of the OS. The network composed of HS and the SDN controller is the controlled plant. Performance characteristics are monitored, and fed back to the GO or the DO, which runs the optimization algorithm and feed the results back to the HS by adjusting the states of their control plane.

The representation of the whole AHS with the different systems and components is shown in 5.10. This figure shows where each component is located and whether it executes on the SDN controller or the switches.

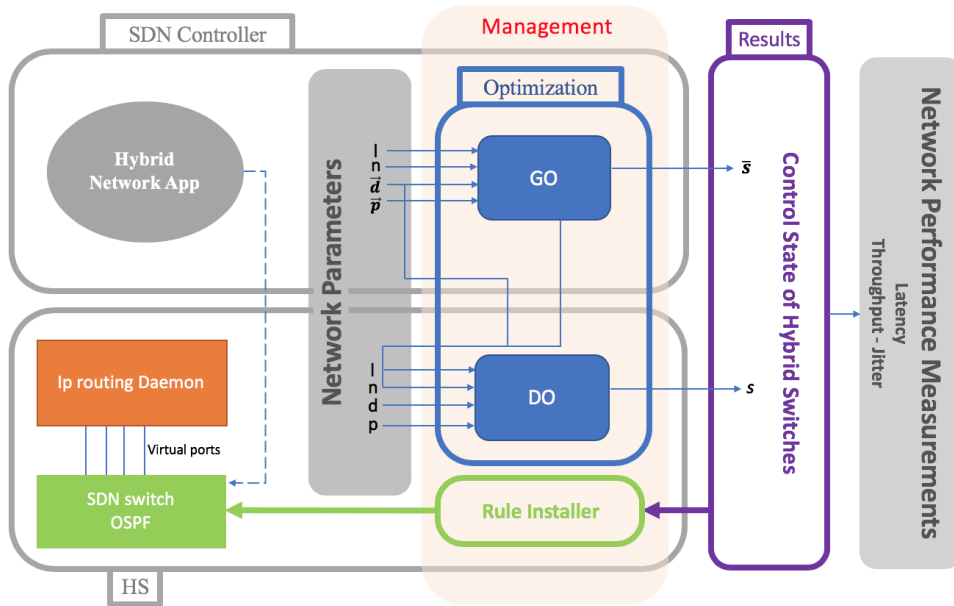


Figure 5.10: AHS sub-systems and components

5.4 Preliminary Optimization Design and Results

The preliminary focused on the DO, and applied the model to telecommunication networks. It also defined the optimization variable as the extent of decentralization of the Agent. This means that the model allowed the Agent to run both control states and rely on central control with a certain percentage of Agents and use distributed control with the rest of the Agents. In this scenario, the CA is the SDN controller, and the Agents are hybrid switches (HS). These switches support both Centralized and Distributed routing protocols. They can be either controlled by the SDN controller in a centralized manner, or they can operate as regular routers and make their own routing decisions in a distributed fashion.

The high-level model is shown in the diagram of Figure 5.11:

- The two outputs correspond to the extent of decentralization in space and

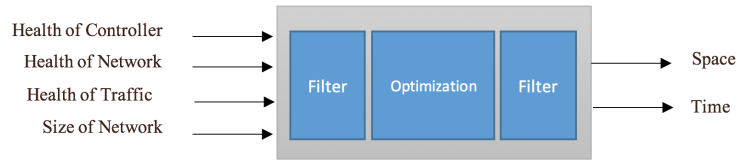


Figure 5.11: Preliminary design high-Level model

time (frequency).

- The system has 4 measured system properties: The Health of the Controller, the Health of the Network, the Health of the Traffic, and the Network Size, which correspond respectively to the CA response time, the stability of the environment, the uncertainty in the environment, and the size of the environment, in the general model. These parameters are explained and translated into network specific measurement in the following sections.

The system we considered includes different time dynamics. It comprises components that operate in different timescales, and at different costs. In fact, the input variables, which are read from sensors, change fast and sampling their values is relatively not expensive. However, controlling the output variables, and shifting the network nodes from centralized to distributed and vice-versa is expensive both in time and overhead. The fast change in the input variables will not translate as quickly on the output side. Consequently, care should be given to these variations, otherwise the stability of the system will be severely affected. We envision filter blocks before and after the optimization block to smoothen out the measured and controllable values (Fig. 1). These filters will also make the system robust to outliers, and prevent the system from directly changing states when it receives an abnormal reading from a sensor.

In this preliminary model, we also defined performance to serve three basic

metrics: availability, reliability and scalability. Consequently, the control rules that we follow aim to maximize one or more of these metrics. Since it is easier to measure delays and costs in the network, we minimize the cost of not achieving performance instead. This is equivalent to minimizing the performance cost incurred by unhealthy controller, network, and traffic, and by the network size.

5.4.1 Optimization Problem Formulation

The system environment is assumed to be an overlay network. The overlay network is modelled by a complete graph (where each node is connected to all other nodes) $G(V,E)$, where V is the set of HS's and E is the set of links (tunnels). Let r_s denote the degree of distributedness in space of the HS s . It represents the amount of control information received from peer HS's over the total amount of control information received. In the overlay network, this percentage can be subsequently translated into the number of peers that the HS can listen to and exchange information with using graph theory. When r_s is 0, the HS is fully centralized, and when r_s is one, the HS is fully distributed.

$$r_s = \frac{\text{ControlPacketsfromDistributedSystem}}{\text{TotalControlpackets}} \quad (5.1)$$

Let f_s denote the degree of distributedness in time of HS s . It represents the normalized frequency per event at which the HS communicates with its peers,

$$f_s = \frac{f_{s,\text{dist}}}{f_{s,\text{max}}} \quad (5.2)$$

where $f_{s,\text{dist}}$ represents the frequency of talking to peers, and $f_{s,\text{max}}$ the maximum bound of $f_{s,\text{dist}}$, which corresponds to maximum frequency at which the network nodes can process messages. Let f'_s denote the degree of centralization

in frequency. It represents the normalized frequency per event at which the HS communicates with the SDN controller, normalized between 0 and 1.

$$f'_s = \frac{f'_{s,\text{cent}}}{f'_{s,\text{max}}} \quad (5.3)$$

where $f'_{s,\text{cent}}$ represents the frequency of talking to the central controller and $f'_{s,\text{max}}$ the maximum bound of $f'_{s,\text{cent}}$. $f'_{s,\text{max}}$ corresponds to the maximum frequency that the server can handle. f_s and f'_s move in opposite directions. When $f_s = 0$, the node is fully centralized, because it is not talking to peers at all. The bigger the f_s , the more decentralized the HS is, since it will be communicating less with the controller.

For each node s , let \mathbf{d}_s represent the measured delay in controller response, and \mathbf{p}_s the probability of unknown packets as seen by s . We also define the parameter \mathbf{l} to represent the rate of topology change observed in the network, and the parameter \mathbf{n} to represent the size of the network. \mathbf{l} and \mathbf{n} are communicated to the switches by the controller. The algorithm is invoked at every T_o period on each HS. For each measured parameter in 5.11, we define an incurred cost. These cost sub-functions are in line with the cost functions defined in the general model, and justified below:

Health of the Controller Cost

The health of the controller (HC) is evaluated by its responsiveness. The associated cost is measured by the delay it takes the controller to reply to the HS's control message (eg. Reply to a Packet-in message).

Control rule 1: When the controller delay increases, instruct the switch to rely less on the controller and go more distributed, to prevent the central control

plane from hindering network Availability.

Rationale: Similar to the situation in political science, when the central authority fails, power is transferred to state governments [33]. The delay between controller and HS indicates that the controller is either overloaded, down, or the link connecting them is malfunctioning. In networks, the central control plane becomes a bottleneck, and excessive reliance on it will incur operation downtime. In this case, the HS should read more control information from the distributed control plane because the central control plane is not able to provide the HS with timely information. Therefore, the distributedness in space should increase, thus r_s should increase. f_s should also increase because the HS should communicate less with a non-responding central control plane. Consequently, to enforce the above behavior, the cost function is proportional to the measured delay and inversely proportional to r_s and f_s . The variable β_1 bounds the controller health cost and keeps it from tending to infinity when the system is in the centralized state (i.e. r_s or f_s are equal to 0). β_1 is a parameter chosen by design: its effect limits the cost to $\frac{1}{\beta_1} \cdot \frac{d}{d_{\max}}$ when r_s or f_s are equal to 0. The cost is defined as follows, where d_s is the delay to controller measured at the HS and d_{\max} is the maximum allowed delay:

$$\text{Cost}_{\text{HC}} = \frac{d_s}{d_{\max}} \cdot \frac{1}{r_s \cdot f_s + \beta_1} \quad (5.4)$$

Health of the Network Cost

The health of network (HN) is evaluated by its stability. The associated cost is measured by the rate of topology changes in the network. This rate is computed by the controller using traffic statistics collected from the network devices, such as the port status message sent by the switch to indicate a change in the

corresponding port status[139]. For example, Opendaylight includes a statistics manager responsible for gathering network information [140].

Control rule 2: When the rate of topology changes increases, instruct the switch to rely on the controller and go more centralized, to ensure higher network Reliability.

Rationale: Similar to the political model, in times of crisis, the control is shifted to the central power. When compared to networks, the SDN controller has better reaction to topology changes. In fact, the controller is aware of the network topology because it implements many functions like topology manager, and continuously updates a Network Information Database (NIB), that keeps track of information about the state of the network such as host locations, link capacities, the traffic matrix, etc. Subsequently, the central control plane can handle frequent topological changes thanks to the network-wide visibility [141]. Therefore, when the topology change rate increases, the switch should be more centralized in space, so r_s should decrease to allow it to rely more on the controller than on its peers. Also, f_s should decrease to allow the switch to talk more frequently to the controller. Consequently, the cost function is proportional to the measured topology change rate and to r_s and f_s , and defined as follows, where l is the rate of topology change and l_{max} is the max allowed rate of topology change:

$$\text{Cost}_{HN} = \frac{l}{l_{max}} \cdot r_s \cdot f_s \quad (5.5)$$

Health of Traffic Cost

The health of traffic (HT) is evaluated by the characteristics of the traffic; known, unknown or abnormal. The associated cost is measured by the rate of unclassified packets, or the rate of packets that the switch receives to which it does not have

an associated flow entry in its table.

Control rule 3: When the rate of unclassified packets increases, the switch should rely more on the central control plane, providing thus improved network Reliability.

Rationale: In fact, since the central control plane has full knowledge of the network, then, given a certain traffic pattern that is unknown to the HS, the HS can benefit from the knowledge of the controller rather than drop the packet, therefore increasing reliability. The packet-in action allows the switch to send an unclassified packet to the controller for further inspection and processing [142]. Consequently, the HS should become less distributed in space and r_s should decrease. Also, exchanging control information with the controller more frequently will help reducing the probability of unclassified packets because the flow-rules will remain up to date, which will consequently reduce the cost of unknown packets. Accordingly, f_s should decrease. The cost is proportional to the unknown packets rate, and to r_s and f_s , and defined as follows, where p_s is the rate of unknown packets, and p_{\max} is the maximum allowed rate of unknown packets:

$$\text{Cost}_{\text{HT}} = \frac{p_s}{p_{\max}} \cdot r_s \cdot f_s \quad (5.6)$$

Network Size

The network size (NS) can be inferred from the size of the routing table, or directly obtained from the controller [141].

Control rule 4: when the size of the network increases, the HS is better off being more distributed. distributed algorithms provide more Scalability in this case.

Rationale: Large networks perform better when the intelligence is distributed

across the system. Consequently, SDN's most important drawback was the lack of scalability in increasing network sizes [143]. Although many techniques were presented to address and account for the scalability issue in SDN, the problem remains a bottleneck. In fact, the issue is not just about processing power of the control plane, it also concerns the increased overhead in task scheduling, increased contention and many other factors that degrade the performance of the central control with an increased number of switches [133]. The cost is thus affected by network size, and inversely proportional to r_s and f_s . Similar to β_1 , the variable β_2 bounds the network size cost and keeps it from tending to infinity when the system is in the centralized state (i.e. r_s or f_s are equal to 0). It limits the cost to $\frac{1}{\beta_2} \cdot \frac{n}{n_{\max}}$ when r_s or f_s are equal to 0. The cost is defined as follows, where n is the network size and n_{\max} is the maximum network size:

$$\text{Cost}_{\text{HC}} = \frac{n}{n_{\max}} \cdot \frac{1}{r_s \cdot f_s + \beta_2} \quad (5.7)$$

Cost_{HN} and Cost_{HT} values range from 0 to 1. Consequently, to keep all four costs within that same range and remove any potential bias, we set β_1 and β_2 to 1. Under this setting, Cost_{HC} and Cost_{NS} will also vary between 0 and 1. θ_s is the total performance cost at HS s , defined as the weighted sum of the above four cost sub-functions:

$$\theta_s = \alpha_1 \cdot C_{s,\text{HC}} + \alpha_2 \cdot C_{s,\text{HN}} + \alpha_3 \cdot C_{s,\text{HT}} + \alpha_4 \cdot C_{s,\text{NS}} \quad (5.8)$$

The weights are chosen by design given the emphasis that the network operator wants to assign to each input on overall performance. Additionally, we set two constraints on the optimization to ensure feasible solutions:

$$p_s(1 - r_s \cdot f_s) < \frac{1}{n} \cdot C \quad (5.9)$$

$$r_s, f_s \in [0, 1] \quad (5.10)$$

Constraint 5.9 bounds the unknown packets rate sent to the controller (the left-hand side) by a maximum value C/n (assuming equal distribution of capacity among HS's) beyond which the switch goes back to a more distributed state, to mitigate the effect of an eventual attack (for eg. DoS on the controller). Equation 5.10 indicates that r_s and f_s are real values in the interval $[0, 1]$. The variables and parameters are summarized in Table 5.1. The optimization problem is defined as:

$$\min_{r_s, f_s \in [0, 1]} \theta_s \quad \forall s \in V$$

subject to:

$$p_s(1 - r_s \cdot f_s) < \frac{1}{n} \cdot C$$

$$r_s, f_s \in [0, 1]$$

where

$$\theta_s = \alpha_1 \cdot C_{s,HC} + \alpha_2 \cdot C_{s,HN} + \alpha_3 \cdot C_{s,HT} + \alpha_4 \cdot C_{s,NS}$$

$$\text{Cost}_{HC} = \frac{d_s}{d_{\max}} \cdot \frac{1}{r_s \cdot f_s + \beta_1}$$

$$\text{Cost}_{HN} = \frac{l}{l_{\max}} \cdot r_s \cdot f_s$$

$$\text{Cost}_{HT} = \frac{p_s}{p_{\max}} \cdot r_s \cdot f_s$$

Variables	
s	A Hybrid Switch in the overlay network
r_s	Extent of decentralization in space of s
f_s	Extent of decentralization in frequency of s
Parameters	
d_s	Delay measured at HS s in ms
d_{\max}	Max allowed d in ms
l	Rate of topology change in changes/s
l_{\max}	Max allowed rate of topology change in changes/s
p_s	Rate of unknown packets at HS s in packets/s
p_{\max}	Max. allowed rate of unknown packets in packets/s
n	Network size
n_{\max}	Maximum network size
C	Capacity of SDN controller
α_1	Controller delay effect on total performance
α_2	Network stability effect on total performance
α_3	Traffic effect on total performance
α_4	Network size effect on total performance

Table 5.1: Variables and parameters

$$\text{Cost}_{\text{HC}} = \frac{n}{n_{\max}} \cdot \frac{1}{r_s \cdot f_s + \beta_2}$$

5.4.2 Modeling of Input Parameters

To simulate the behavior of the system, we modeled each of the input parameters.

Rate of Unknown Packets

The rate of packet arrival at a given switch is modeled by a Poisson distribution [144], with expected value of λ . Consequently, the rate of unknown packets is modeled as:

$$E[\text{Rate}] = \lambda \times p \quad (5.11)$$

Where λ is the average arrival rate in the switch which follows a Poisson Process (the arrivals in different switches are independent), and p is the probability

of unknown packets.

Controller Delay

Given that the rates of arrivals at the switches are independent, if the network is composed of N active switches, then, the rate of packets-in messages seen at the controller is given by the sum of the rates of unknown packets at each switch. The rate of arrival is given by:

$$\lambda_c = \int_1^N \lambda_i \times p_i \quad (5.12)$$

Since the rate of arrival at each switch follows a Poisson distribution with parameter λ_i , the rate of packet-in at the controller also follows a Poisson distribution with parameter λ_c (composition algorithm). Additionally, in probability, the time spent between events in a Poisson distribution follows the exponential distribution [119]. Consequently, the time it takes for the controller to process the packet-in messages is given by the exponential distribution with parameter μ_c . As a result, processing packets-in at the controller can be modeled as an M/M/1 queue. The average time a packet spends in the controller, including the time of waiting in the queue and processing time, is defined by [145] as follows:

$$E[T_c] = \frac{1}{\mu_c - \lambda_c} \quad (5.13)$$

We defined the controller delay to be the delay of controller response as seen by the switch. Consequently, the average controller delay is the sum of the time the packet spends at the controller added to the round-trip travel time from the switch to the controller, modeled by a normal distribution. The average controller delay is thus given by:

$$E[D_c] = E[T_c] + E[RTT] \quad (5.14)$$

Topology Change

We modeled the topology change as the sum of four rates (assuming each of the four types of events happens independently of the other), as shown below:

- Link failure and link creation: This corresponds to the probability of one link failing, modeled as an exponential failure density distribution with constant failure rate [146].
 - Reliability function (No failure occurs before time t):

$$R(t) = e^{-\lambda t} \quad (5.15)$$

Where $\lambda(t)$ is the failure rate (constant).

- Exponential failure density function:

$$f(t) = \lambda e^{-\lambda t} \quad (5.16)$$

With $t \geq 0$, With mean or expected value of

$$E(x) = \frac{1}{\lambda} \quad (5.17)$$

By extrapolation, link creation is also modeled with the exponential distribution.

- Node failure: Node Failure can be also modeled by the exponential failure. However, switches have a lifetime and they undergo wear-out with time.

Consequently, the Weibull distribution can be used to account for this effect [147]. Probability density function

$$f(t) = \frac{k}{\lambda} \frac{t^{k-1}}{\lambda} e^{-\left(\frac{t}{\lambda}\right)^k} \quad (5.18)$$

With $t \geq 0$, With mean or expected value of

$$E(x) = \lambda \Gamma\left(1 + \frac{1}{k}\right) \quad (5.19)$$

Where Γ is the gamma function, with a shape parameter k and a scale parameter $\lambda > 0$.

- Node creation: Assuming that node creation corresponds to a pure birth process, it is modeled as a Poisson process with rate of λ [148]. Death process has been modeled previously with the Weibull distribution.

Size of Network

The size of the network is an integer number ranging from 1 to N where N is the maximum number of switches that we define. Consequently, it is modeled with a uniform distribution over the interval $[1, N]$ as follows:

$$E(x) = \frac{1}{2}(1 + N) \quad (5.20)$$

Input parameters models are summarized in Table 5.2.

Input Modeling	Distribution	Expression
1. Rate of Unknown Packets	Poisson	Rate = $\lambda \times p$
2. Controller Delay		$E[D_c] = E[T_c] + E[RTT]$
a. Queue Time	M/M/1 Queue	$E[T_c] = \frac{1}{\mu_c - \lambda_c}$
b. Round Trip Time	Normal Distribution	$E[RTT]$
3. Topology Change		
a. Link Failure	Exponential Distribution	$E(X) = \frac{1}{\lambda}$
b. Link Creation	Exponential Distribution	$E(X) = \frac{1}{\lambda}$
c. Node Failure	Weibull Distribution	$E(x) = \lambda \Gamma(1 + \frac{1}{k})$
d. Node Arrival	Poisson	$E(X) = \lambda$
4. Network Size	Uniform Distribution	$E(x) = \frac{1}{2}(1 + N)$

Table 5.2: Input modeling

5.4.3 Results

We tested our optimization problem to verify its behavior under different network conditions. We simulated varying network conditions using the models defined in Table 5.2. The built-in Matlab function *fmincon*, which implements the interior point algorithm, was used to solve the optimization problem. Figure 5.12 shows the performance of our optimization problem with respect to a fully centralized network and a fully distributed one, with changing network conditions over time by varying the means of the different distributions between 0 and 1 with a step of 0.1. As expected, the optimized scenario (red line) has the least cost.

Behavior of the Optimization System

Figures 5.13 to 5.20 show the system's behavior for varying network conditions. For each 3D plot, we fix 2 input variables to either their minimum or maximum value, and observe the effect of the variations of the other two variables on the decision. In Figure 5.13, we set the mean of the delay to controller variable to its minimum value and the rate of topology change to its maximum value. These are conditions favorable to centralization. The graph shows that the resulting

system is always centralized regardless of the value of the network size or the rate of unknown traffic. A slight increase towards decentralization occurs when the rate of unknown traffic is minimal and the network size is big (which is in line with our control rules). In Figure 5.14, the delay to controller is set to its maximum and the rate of topology change to its minimum value (conditions favoring the distributed state). In this case, When the rate of unknown traffic is low, the system is distributed. It moves towards to the centralized state as the rate increases (as expected). The system moves to the fully centralized state when the network size is small and the rate of unknown packets is at its maximum. Figure 5.15 shows the results when the delay to controller is at its minimum value and the rate of unknown packets is at its minimum. This condition doesn't favor one state over the other. However, we notice from the plot that the system tends to the decentralized state for most of the values of the network size and the rate of topology change, except when either has a low value. When the rate of topology change is at its maximum, the system moves quickly to the centralized state to give the SDN controller full control of the unstable network. Figure 5.16 shows the results when the delay to controller is at its minimum value and the rate of unknown packets is at its maximum. The system as such favors the centralized state for all values of the other two input parameters. In Figure 5.17, the rate of topology change and the rate of unknown packets are set to their minimal values. Results show that the optimization returns a fully distributed network for all values of network size and delay to controller. When the delay to controller and the network size are minimal, the system is slightly less distributed. Figure 5.18 represents the conditions when the rate of topology change and the rate of unknown packets are at their maximum. The system is centralized for all values of the other two input parameters. Figure 5.19 shows optimization results when

the delay to controller is minimum, and the network size is the smallest. In this case, we expect the values of r and f to be 0, because when the delay to controller and network size are small, centralization is advantageous. When the rate of unknown traffic is small, as the topology change rate starts decreasing, the system starts becoming more distributed, which means that the system becomes more decentralized. When the topology change rate and unclassified packets are at their minimum, the system becomes very distributed (almost fully distributed). Figure 5.20 shows optimization results when the delay to controller is maximum, and the network size is the biggest. When the rate of unknown traffic is low the system is fully distributed, except for high values of the rate of topology change where the system became less distributed. The system becomes the most centralized when the rate of unknown traffic and the rate of topology change are at their maximum.

Effect of a Single Input Change on Optimal Decision

The graphs in Figure 5.21 show the extent of centralization required under different network conditions, in three scenarios:

1. When the network conditions are favorable for a fully distributed system (red line)
2. When the network conditions are favorable for fully centralized system (blue line)
3. When the network conditions are equally favorable to either forms (green line)

Figure 5.21 shows results when fixing the means of 3 input variables and showing the effect of the fourth input variable on the optimization output. We

notice that when the conditions are favorable for decentralization (scenario 1), the system always tends towards fully distributed control, for all values of the fourth input variable. Alternatively, in scenario 2, the system is always centralized for all values of the fourth input variable. In Figure 5.21.a, when the network is tested under scenario 3, an increase in the delay to controller moves the control plane more to the distributed state (consistent with control rule 1). The same behavior is observed in Figure 5.21.c, where an increase in network size causes the network control to go towards a distributed state (control rule 4). In Figure 5.21.b, we notice that the rate of topology change increases, the network goes more distributed (control rule 2). As for the rate of unknown number of packets (Figure 5.21.d), the increase in the rate causes the network to become more centralized, as expected (control rule 3).

Effect of the Constraint

We performed another test where we varied the capacity of the controller to see the effect of the constraint on the optimization problem. The network was tested in scenarios 1 and 2. Figure 5.23.a shows the results when increasing the rate of topology change, while Figure 5.23.b shows the results when increasing the rate of unknown traffic, in conditions that are favorable to centralization (scenario 2). We notice that as the controller capacity decreases, the system becomes more distributed. This is expected because the smaller the capacity of the controller, the switches should depend less on this latter to avoid creating a bottleneck.

Analysis

Results showed that our optimal system has lower cost than a fully distributed network or a fully centralized one, running in the same network conditions (Figure

5.12). Additionally, the variations of the level of centralization vs. distributedness with respect to changing network conditions correspond to the control model that we had specified based on guidelines in previous sections. Actually, a minimum delay to controller favors centralization. In this case, if any of the rate of topology change, the rate of unknown packets, or the network size, favors centralization (i.e. maximum rate of topology change-Figure 5.13, maximum rate of unknown packets - Figure 5.16, or minimum network size-Figure 5.19), the system basically tends to full centralization, for all values of the other two system inputs. In fact, when the controller is highly responsive, it is better to rely more on this latter and take advantage of its global knowledge. However, if the rate of unknown packets favors decentralization (i.e. min rate of unknown packets - Figure 5.15), the system is better off distributed. In this case, given that the rate of packets is minimum, the traffic is known to the node and no further information is needed from the controller, even if this latter is highly responsive. Consequently, the system will be distributed, except in two cases: when the topology changes frequently, in which case the node would require the global topology knowledge from the controller, or when the network size is small. Furthermore, if the topology is frequently changing and the traffic is unfamiliar to the node (i.e. the rate of unknown packets is high) (Figure 5.18), the node is expected to rely more on the controller, regardless of the speed of its response. In fact, neither the topology nor traffic are known to the node, thus the controller guidance is beneficial. In our model, the delay to controller is bounded by a maximum delay and a maximum controller capacity C , which keeps the controller from overloading and alternatively limits the controller response delay. Alternatively, if the changes in topology are very rare, and the traffic is known to the network (Figure 5.17), the system is decentralized for all values of delay to controller and network size. In

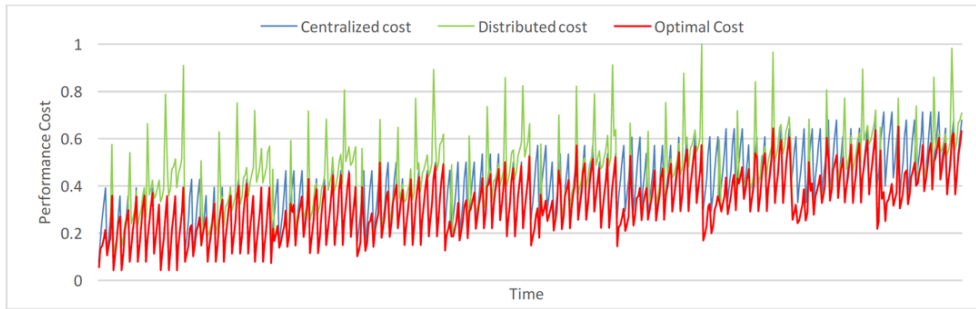


Figure 5.12: Performance of our system in comparison with fully centralized and fully distributed settings

fact, the controller’s help is not needed in this case, and its responsiveness does not affect the distributed model; the network size does not affect the distributed model either, because this latter supports scalability. Finally, if the controller is not responsive and the network is large (Figure 5.20), then the network is distributed for known traffic and for infrequent topology changes. However, as the traffic becomes unfamiliar and the topology starts changing frequently, the network is better off relying on the controller despite of its slow response, mainly because the network node needs topology and the traffic from the omniscient controller.

5.5 Final Model Formulation

The solutions of the preliminary control model resulted in Hybrid Switches that use central and distributed control planes at the same time, i.e. the HS’s rely on the SDN to communicate with some parts of the network, while use OSPF to communicate to the rest of the network. Another approach is to cast the HS’s to be either centralized or distributed. The resulting network as such would be hybrid, whereas each HS by itself is either centralized or distributed.

This drove a variation of the previously defined distributed optimization (DO)

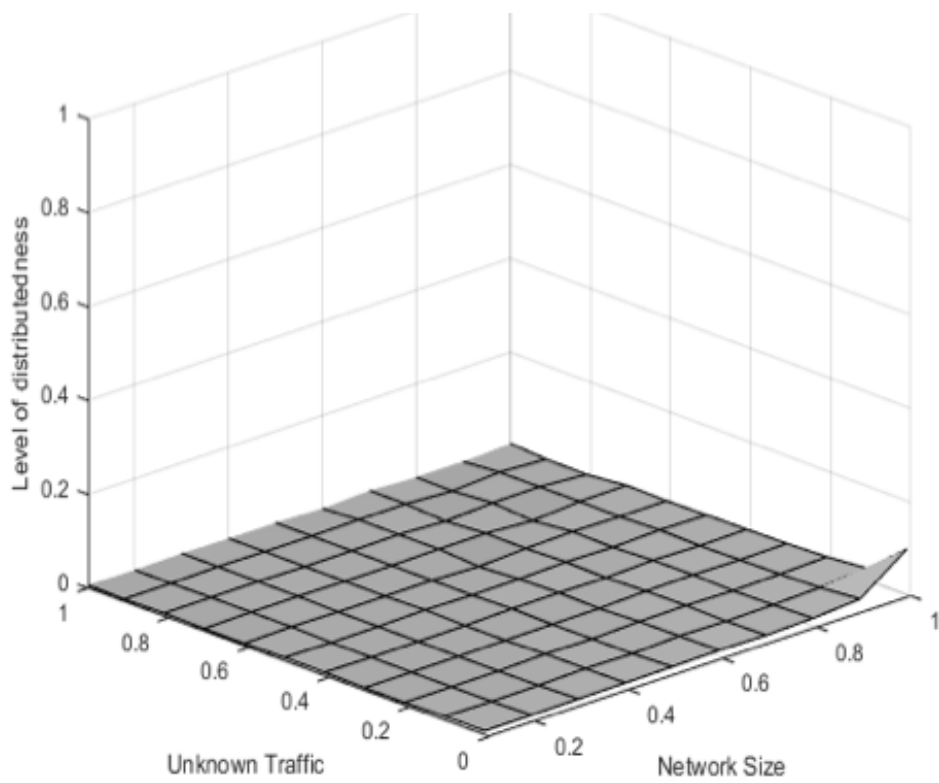


Figure 5.13: Min delay to controller - max rate of topology change

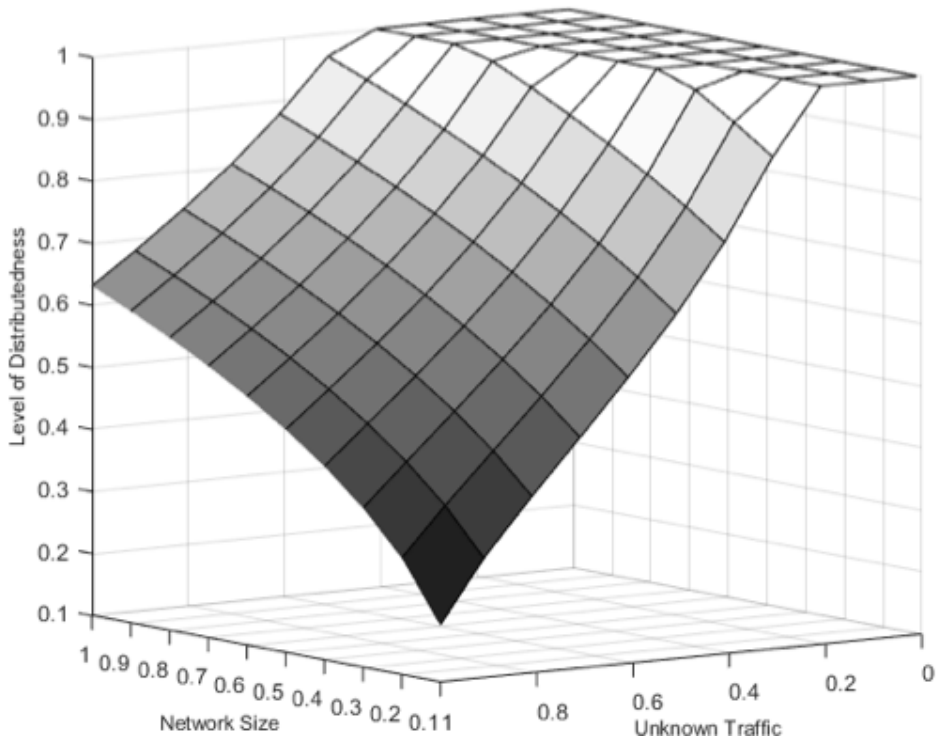


Figure 5.14: Max delay to controller - min rate of topology change

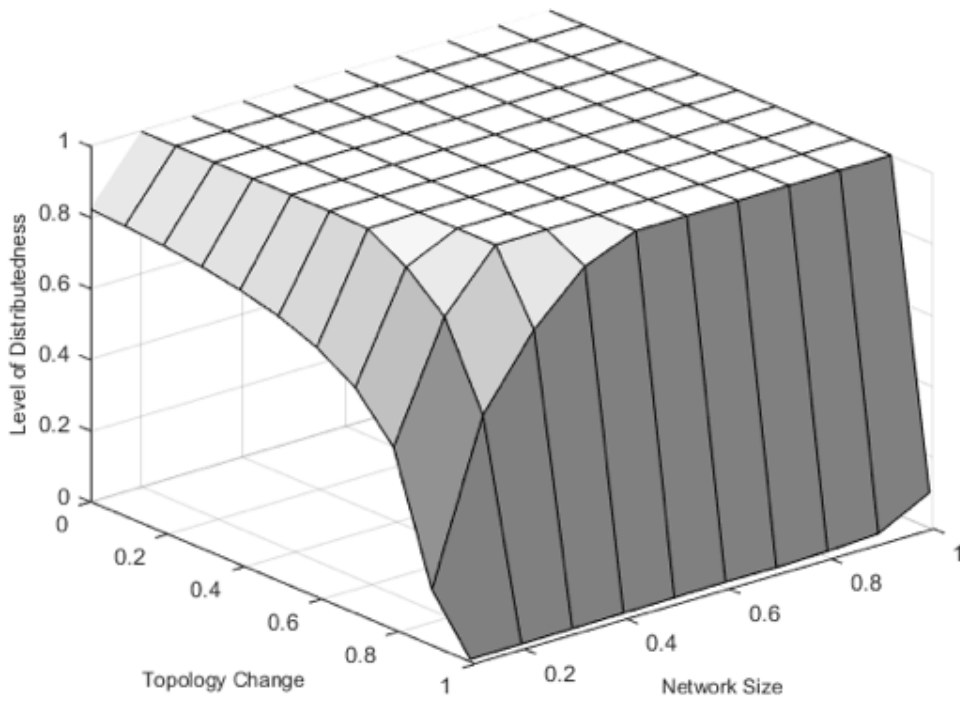


Figure 5.15: Min delay to controller - min rate of unknown packets

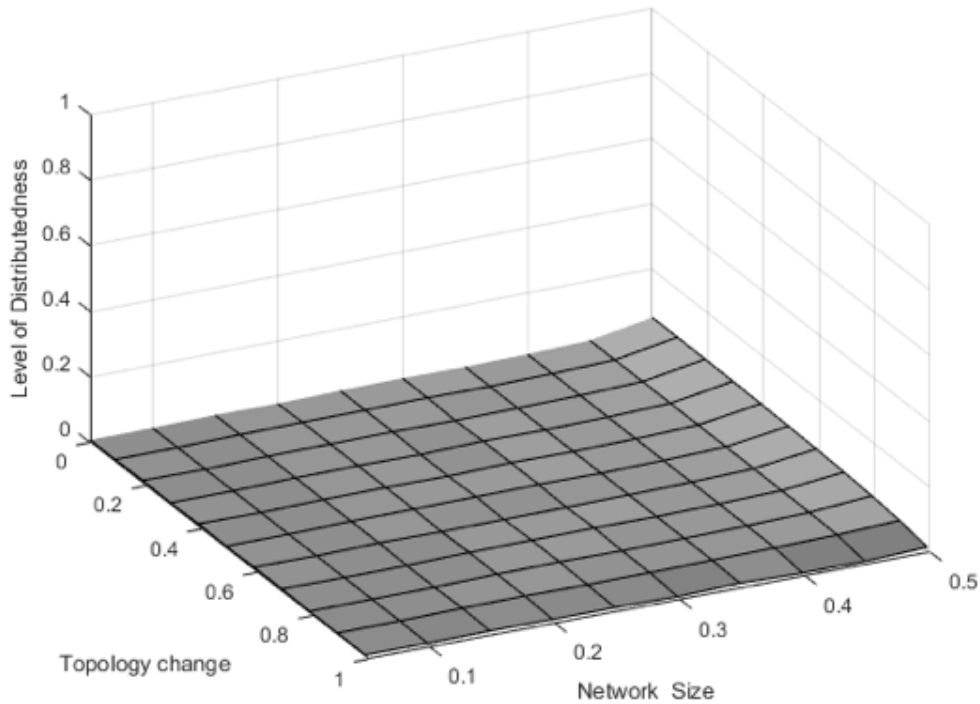


Figure 5.16: Min delay to controller - max rate of unknown packets

system in section 5.4. This optimization follow the same rules as the General Optimization system.

5.5.1 DO problem Formulation

Since this DO optimization problem decides on the control state of the HS, we define the system variable s differently than the variable of the preliminary model r . In this formulation, s the probability of the state of the control plane of the HS. When $s = 0$, the HS is fully centralized, and when $s = 1$, the HS relies completely on the distributed control plane. As in the preliminary model, the supervised variables are defined for networks as follows:

1. The controller response time 'd' to a message received from a switch. It represents the controller's availability, reachability or responsiveness.

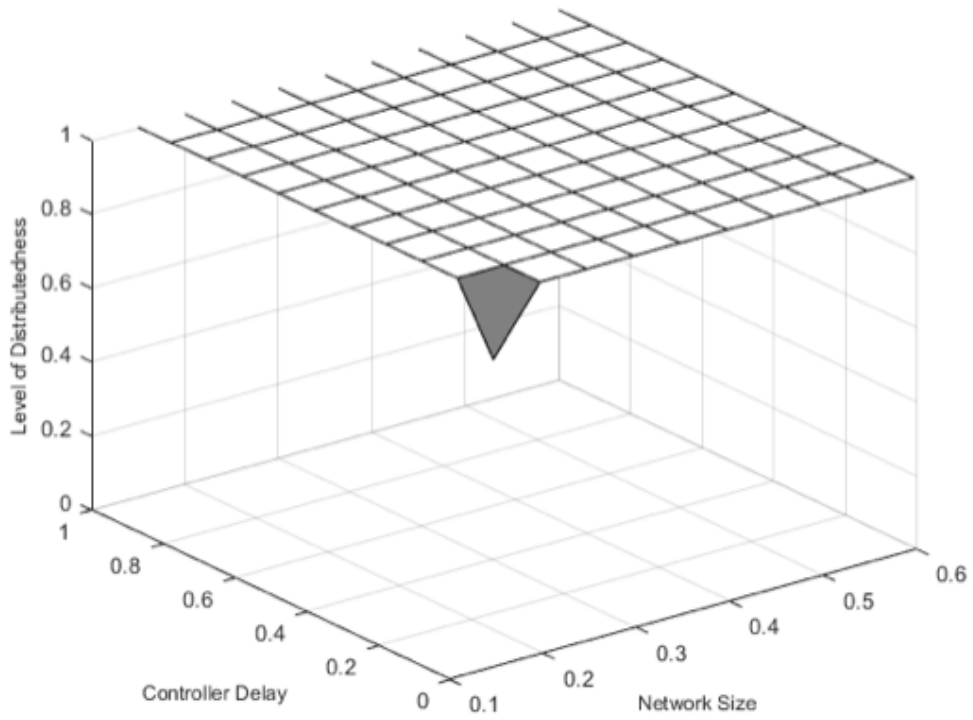


Figure 5.17: Min rate of topology change - min rate of unknown packets

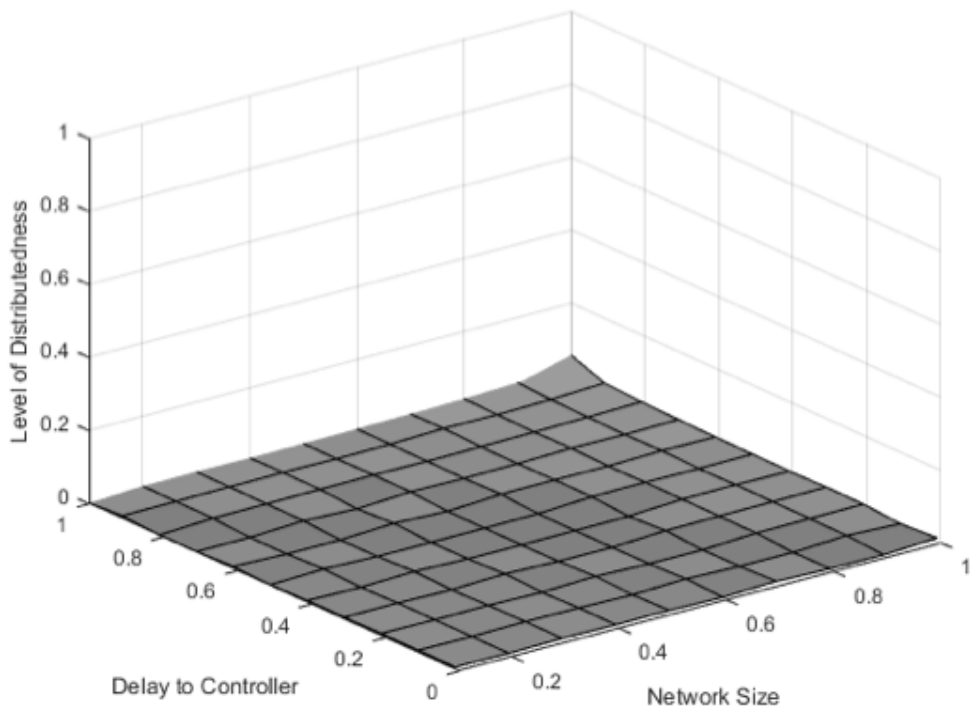


Figure 5.18: Max rate of topology change - max rate of unknown packets

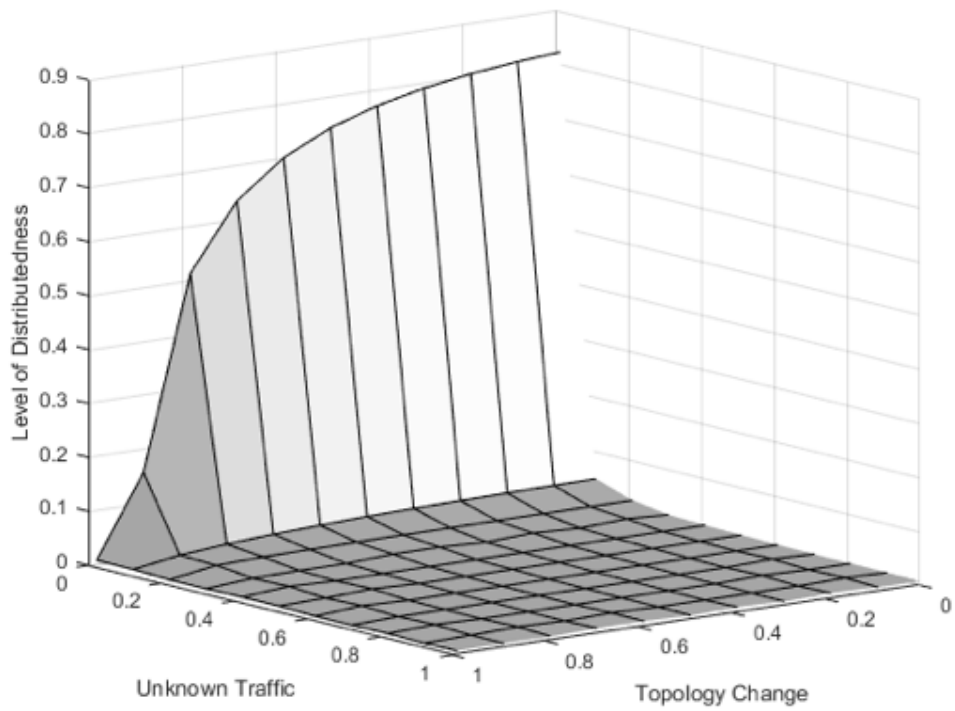


Figure 5.19: Min delay to controller – min network size

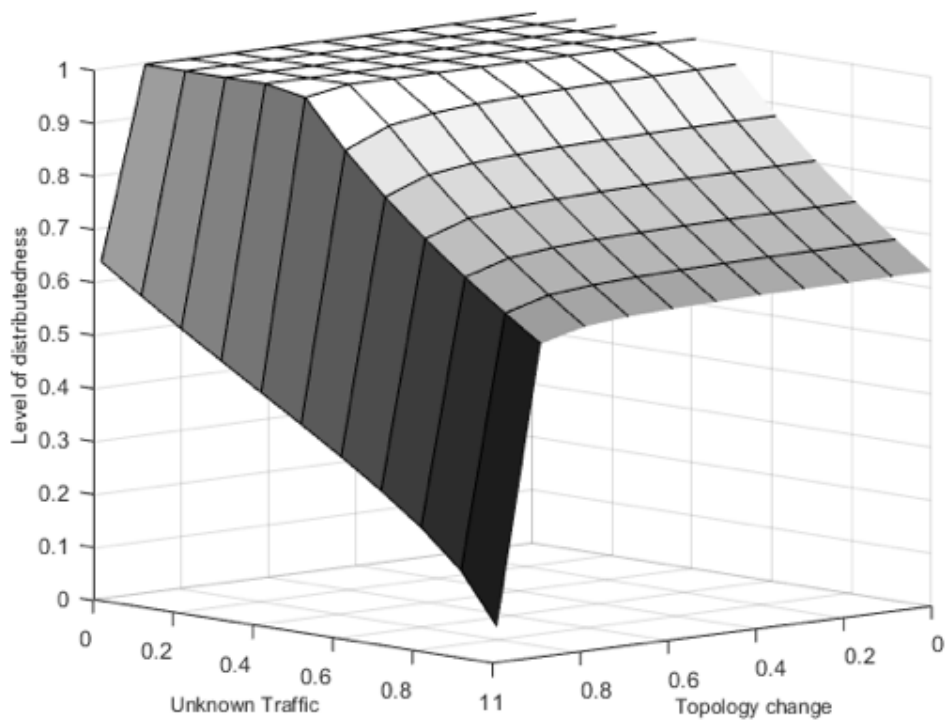


Figure 5.20: Max delay to controller – max network size

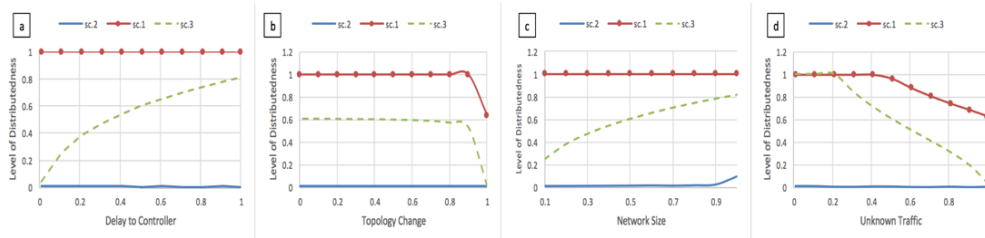


Figure 5.21: Optimization solution for different network conditions

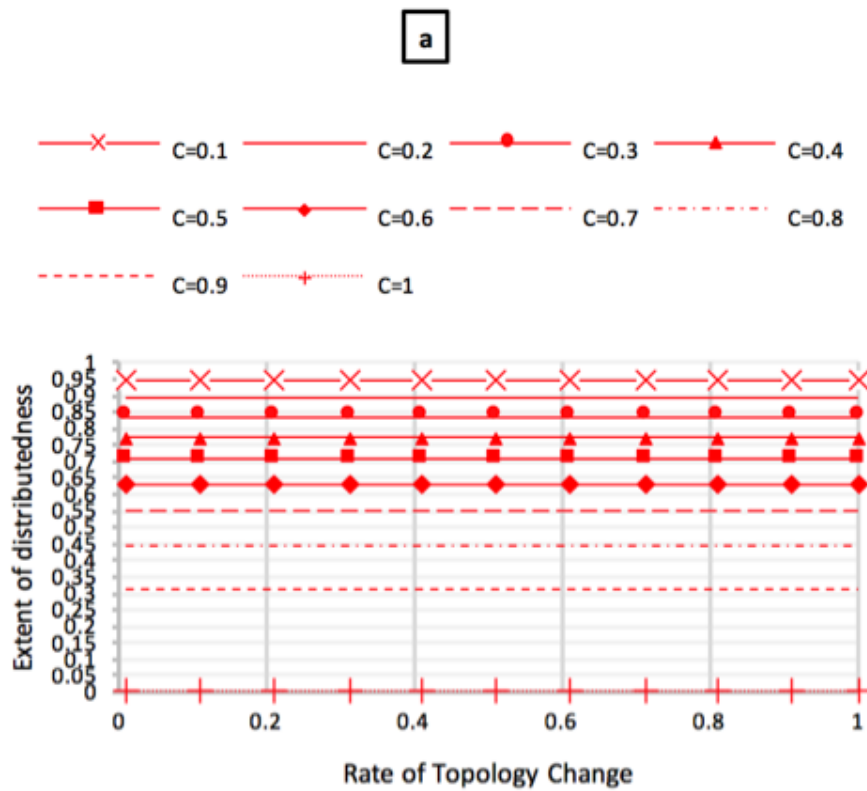


Figure 5.22: Effect of the condition on the optimization results - a

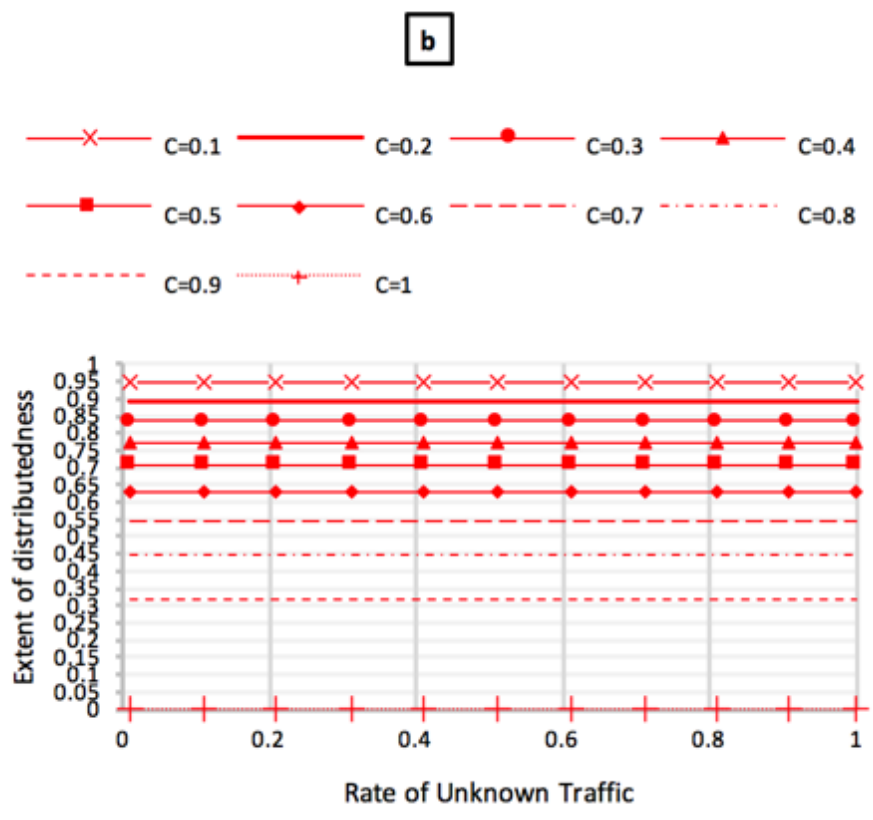


Figure 5.23: Effect of the condition on the optimization results - b

Variables	
s	Probability of state of the HS's control plane
Parameters	
n	Size of the network (Normalized)
d	SDN controller delay (Normalized)
t	Rate of topology changes (Normalized)
p	Number of unknown packets (Normalized)
g_i	Cost functions
h_i	Mapping functions
α_i	Weighting factors
θ_A	Robustness Criterion at the HS
θ_{CA}	Global Robustness Criterion at the SDN controller

Table 5.3: Variables and parameters

2. The rate of topology change 't'.
3. The Count of unknown packets 'p' ("Packet In" to the controller) which represent the amount of unclassified packets in the traffic.
4. The network size 'n', which corresponds to $n = n_n + n_l$, where n_n is the number of network nodes and n_l is the size of network links.

The system variables are defined in table 5.3

The control rules defined previously are applied to this model as well. The below elaborates on the techniques used to formulate the sub-cost functions for the optimization problem. For all 4 parameters, we define the expected response of the optimization problem at the boundaries.

Controller Response Time

If the SDN controller is non-responsive, or if the delay between the controller and the switcher is large, the switcher should go distributed to prevent the central control plane from hindering network Availability. At the boundaries, when:

$$\begin{cases} \mathbf{d} = 0 \rightarrow \mathbf{s} = 0 \\ \mathbf{d} = 1 \rightarrow \mathbf{s} = 1 \end{cases} \quad (5.21)$$

the mapping from \mathbf{d} to \mathbf{s} can be linearly defined as $\mathbf{s} = \mathbf{d}$.

Let \mathbf{g}_1 be the cost function that relates \mathbf{s} to \mathbf{d} . In order to maximize \mathbf{g}_1 , solve $\mathbf{g}'_1(\mathbf{s}) = 0$. Given the boundaries above (eq. 5.21), we deduce that a solution that maximize \mathbf{g}'_1 is $\mathbf{s} = \mathbf{d}$. In other words, we want to solve for \mathbf{s} when \mathbf{g}'_1 is set to 0, such that the above mapping between \mathbf{s} and \mathbf{d} becomes a solution.

$$\mathbf{g}'_1(\mathbf{s}) = 0 \Rightarrow \mathbf{s} = \mathbf{d} \Rightarrow \mathbf{g}'_1(\mathbf{s}) = \mathbf{d} - \mathbf{s} \Rightarrow \mathbf{g}_1(\mathbf{s}) = \mathbf{d}\mathbf{s} - \frac{1}{2}\mathbf{s}^2 \quad (5.22)$$

For a more general solution, the mapping between ‘ \mathbf{s} ’ and ‘ \mathbf{d} ’ could be any function that obeys conditions in 5.21. The mapping can be chosen to be any other function depending on the behavior to enforce in the system. It follows that

$$\mathbf{g}_1(\mathbf{s}) = \mathbf{s} \cdot \mathbf{h}_1(\mathbf{d}) - \frac{1}{2}\mathbf{s}^2$$

where $\mathbf{h}_1(\mathbf{d})$ is the mapping function between ‘ \mathbf{s} ’ and ‘ \mathbf{d} ’.

The Rate of Topology Change

If the topology of the network keeps changing, the central SDN control plane provides a comprehensive view of the network that makes it react better to these changes and ensure higher network Reliability. At the boundaries when

$$\begin{cases} \mathbf{t} = 0 \rightarrow \mathbf{s} = 1 \\ \mathbf{t} = 1 \rightarrow \mathbf{s} = 0 \end{cases} \quad (5.23)$$

the mapping from \mathbf{t} to \mathbf{s} can be linearly defined as $\mathbf{s} = 1 - \mathbf{t}$.

Following the same approach as for g_1 , let g_2 be the cost function that relates s to t . Then to maximize g_2 , find g'_2 such that the above mapping becomes a solution when $g'_2=0$.

$$g'_2(s) = 0 \Rightarrow s = 1 - t \Rightarrow g'_2(s) = 1 - (t + s) \Rightarrow g_2(s) = (1 - t)s - \frac{1}{2}s^2 \quad (5.24)$$

More generally, $g_2(s) = s \cdot h_2(t) - \frac{1}{2}s^2$

Again, we notice that the function g_2 has the same form as the function g_1 with $h_2(t) : s = 1 - t$ (linear model).

Count of Unknown Packets

If the traffic features many packets that are unknown to the switcher, then this latter can make use of the knowledge of the central control plane to avoid dropping packets, providing thus improved network Reliability. At the boundaries when

$$\begin{cases} p = 0 \rightarrow s = 1 \\ p = 1 \rightarrow s = 0 \end{cases} \quad (5.25)$$

the mapping from p to s can be linearly defined as $s = 1-p$.

Similarly, let g_3 be the cost function that relates s to p , then to maximize g_3 , solve for s when its derivative is set to 0, such that the above mapping becomes a solution.

$$g'_3(s) = 0 \Rightarrow s = 1 - p \Rightarrow g'_3(s) = 1 - (p + s) \Rightarrow g_3(s) = (1 - p)s - \frac{1}{2}s^2 \quad (5.26)$$

More generally, $g_3(s) = s \cdot h_3(p) - \frac{1}{2}s^2$

Network Size

As the network size increases, the centralized control plane provided with SDN start becoming a bottleneck; distributed algorithms provide more Scalability in this case. Consequently, at the boundaries when

$$\begin{cases} \mathbf{n} = 0 \rightarrow \mathbf{s} = 0 \\ \mathbf{n} = 1 \rightarrow \mathbf{s} = 1 \end{cases} \quad (5.27)$$

the mapping from \mathbf{n} to \mathbf{s} can be linearly defined as $\mathbf{s} = \mathbf{n}$.

Again, let \mathbf{g}_4 be the cost function that relates \mathbf{s} to \mathbf{n} , then to maximize \mathbf{g}_4 , solve for \mathbf{s} when its derivative is set to 0, such that the above mapping becomes a solution.

$$\mathbf{g}'_4(\mathbf{s}) = 0 \Rightarrow \mathbf{s} = \mathbf{n} \Rightarrow \mathbf{g}'_4(\mathbf{s}) = \mathbf{n} - \mathbf{s} \Rightarrow \mathbf{g}_4(\mathbf{s}) = \mathbf{n}\mathbf{s} - \frac{1}{2}\mathbf{s}^2 \quad (5.28)$$

More generally, $\mathbf{g}_4(\mathbf{s}) = \mathbf{s} \cdot \mathbf{h}_4(\mathbf{n}) - \frac{1}{2}\mathbf{s}^2$

Maximizing Robustness

As previously defined, Robustness is achieved by improved *Availability*, *Reliability*, and *Scalability*. These metrics are in turn defined as the sum of the four cost functions $\mathbf{g}(\mathbf{s})$. Consequently, Robustness is given by:

$$\theta_s(\mathbf{s}) = \alpha_1 \cdot \mathbf{g}_1(\mathbf{s}) + \alpha_2 \cdot \mathbf{g}_2(\mathbf{s}) + \alpha_3 \cdot \mathbf{g}_3(\mathbf{s}) + \alpha_4 \cdot \mathbf{g}_4(\mathbf{s}) \quad (5.29)$$

$$\theta_s(s) = \alpha_1(s \cdot h_1(d) - \frac{1}{2}s^2) + \alpha_2(s \cdot h_2(t) - \frac{1}{2}s^2) + \alpha_3(s \cdot h_3(p) - \frac{1}{2}s^2) + \alpha_4(s \cdot h_4(n) - \frac{1}{2}s^2) \quad (5.30)$$

$$\theta_s(s) = s(\alpha_1 h_1(d) + \alpha_2 h_2(t) + \alpha_3 h_3(p) + \alpha_4 h_4(n)) - \frac{1}{2}s^2(\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) \quad (5.31)$$

$$\theta'_s(s) = (\alpha_1 h_1(d) + \alpha_2 h_2(t) + \alpha_3 h_3(p) + \alpha_4 h_4(n)) - s(\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) \quad (5.32)$$

To maximize Robustness, set $\theta'_s(s) = 0$ to find maximum of $\theta_s(s)$:

$$\theta'_s(s) = 0 \quad (5.33)$$

$$(\alpha_1 h_1(d) + \alpha_2 h_2(t) + \alpha_3 h_3(p) + \alpha_4 h_4(n)) - s(\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) = 0 \quad (5.34)$$

$$\Rightarrow s = \frac{\alpha_1 h_1(d) + \alpha_2 h_2(t) + \alpha_3 h_3(p) + \alpha_4 h_4(n)}{\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4} \quad (5.35)$$

The optimization problem has a closed form solution s which represents the extent of centralization or decentralization of the switcher. This corresponds to the probability of the control state for the switcher.

The DO algorithm is also invoked at every T_0 period on each HS. Alternatively, the HS recomputes its optimization algorithm at any event occurrence advertised

Variables	
s	Probability of state of the HS's control plane
\dot{s}	Percentage of distributedness in the network
Parameters	
n	Size of the network (Normalized)
d	SDN controller delay (Normalized)
\bar{d}	Characteristic value of Controller delay d (Normalized)
t	Rate of topology changes (Normalized)
p	Number of unknown packets (Normalized)
\bar{p}	Characteristic value of Number of unknown packets p (Normalized)
g_i	Cost functions
h_i	Mapping functions
α_i	Weighting factors
θ_A	Robustness Criterion at the HS
θ_{CA}	Global Robustness Criterion at the SDN controller

Table 5.4: Variables and parameters

by its neighbor or by the SDN controller. The T_{HoldOff} timer also applies (refer to section 5.2).

5.5.2 GO Problem Formulation

For the global optimization, the controller solves the same optimization system as the distributed model, however using different definitions for some parameters. In fact, the rate of changes in the network and the network size are already measured globally; consequently, their values are directly fed to the centralized model. However, when it comes to the number of unknown events and the responsiveness delays of the SDN controller, a characteristic value of the parameters values of all HS's is inputted to the global optimizer. The characteristic value can be for eg. the median or the average. Table 5.4 shows the variables introduced in this model added to the previous variables.

Using the closed form solution as for the distributed model, we compute \dot{s} , which corresponds to the percentage of HS's with distributed control plane in the

network, as follows:

$$\dot{s} = \frac{\alpha_1 h_1(\dot{d}) + \alpha_2 h_2(t) + \alpha_3 h_3(\dot{p}) + \alpha_4 h_4(n)}{\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4} \quad (5.36)$$

Where \dot{d} and \dot{p} are the characteristic value of the HS's d and p values respectively.

In this model, the results of the distributed optimization are used to decide which HS's will have the centralized control plane or distributed control plane given the result of the global optimization, as shown in Figure 5.7.

5.5.3 Design Alternatives

The initial design combines GO and DO; however it is worth noting that each of the DO and GO is a stand-alone solution which provides results independently of the other. Consequently, we propose design alternatives as special cases of our proposed technique.

Special case where GO is not available

In this case, each HS decides if it should have centralized or decentralized approach based on the DO algorithm. Since the closed loop solution s of the DO provides a scalar that represents the probability of the state that the HS should be in, we introduce a uniformly distributed random process to obtain the final state of the HS: either centralized ($s = 0$) or decentralized ($s = 1$).

Special case where DO is not available

The SDN controller runs the GO, computes the percentage of distributedness of the system, and instructs the HS's of the state they should be in. Since \dot{s}

provides the percentage of HS's that should have decentralized control, we rely on the responsiveness delay (parameter d) to decide between HS's: the $s\%$ HS's with the highest delay to the SDN controller will be decentralized.

Case where both GO and DO are available: Hybrid Optimization (HO)

In this case, both optimizations are available, but they are not run together. Each optimization will be invoked given the nature of the events that are taking place in the network. In fact, we classify events between local issues and global issues. In the general model, local issues are generally issues that affect one Agent, or a small group of neighboring Agents, whereas Global issues are general issues that affect the welfare of the whole system. To define the network wide issues and the local issues in telecom networks, examples of local issues can be a 'port down', a bad connection to controller, unknown traffic to the node, whereas global issues can be attacks, frequent link drops, huge traffic, critical traffic, etc. We then assign the GO to network wide issues, while we assign DO to local issues. In this scenario, we assume that the controller is always available and has full knowledge and control of the network. The rationale behind this scheme is as follows: if the local issues can be resolved at the HS's level, there is no need to involve the SDN controller in the optimization process. Additionally, results would be faster because we avoid the delay incurred by running two optimizations.

The three design alternatives are also special cases of our proposed bi-level optimization - instead of being alternatives to it. In fact, the first two alternatives provide robustness, where if one optimization fails in the bi-level system, the other automatically takes over the optimization as a failover solution. Also, the third design alternative, when used for events happening within the period T_0 ,

provides efficiency to the bi-level system. In fact, if an event happens and triggers re-computation of the optimization, then based on the nature of the event, only the appropriate optimization executes (DO or GO), instead of rerunning the full bi-level optimization.

5.6 Results and Discussion

5.6.1 DO Testing

We started first by testing the DO. The system is implemented using Matlab, on a network of 20 nodes. Since the closed loop solution s provides a scalar that represents the probability of the state that the HS should be in, we used a uniformly distributed random process to obtain the final state of the HS: either centralized ($s = 0$) or decentralized ($s = 1$). To study the effect of the decision threshold on the result, we define two thresholds: t_1 gives equal chances to both states, whereas t_2 favors the centralized state. To simulate random network conditions, we generated random vectors for the input variables d , t , p , and n , according to equations 5.37 to 5.40.

$$\mathbf{d}_i = \mathbf{c}_1 \times \mathbf{rand} + \mathbf{c}_2 \times \mathbf{d}_{i-1} + \mathbf{c}_3 \times \mathbf{d}_{i-2} \quad (5.37)$$

$$\mathbf{t}_i = \mathbf{c}_1 \times \mathbf{rand} + \mathbf{c}_2 \times \mathbf{t}_{i-1} + \mathbf{c}_3 \times \mathbf{t}_{i-2} \quad (5.38)$$

$$\mathbf{p}_i = \mathbf{c}_1 \times \mathbf{rand} + \mathbf{c}_2 \times \mathbf{p}_{i-1} + \mathbf{c}_3 \times \mathbf{p}_{i-2} \quad (5.39)$$

$$\mathbf{n}_i = \mathbf{c}_1 \times \mathbf{rand} + \mathbf{c}_2 \times \mathbf{n}_{i-1} + \mathbf{c}_3 \times \mathbf{n}_{i-2} \quad (5.40)$$

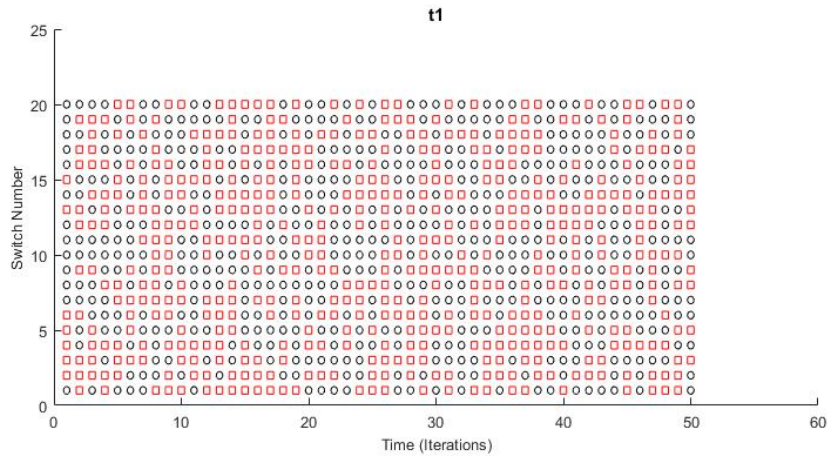


Figure 5.24: Control plane states for DO with t_1 in random conditions

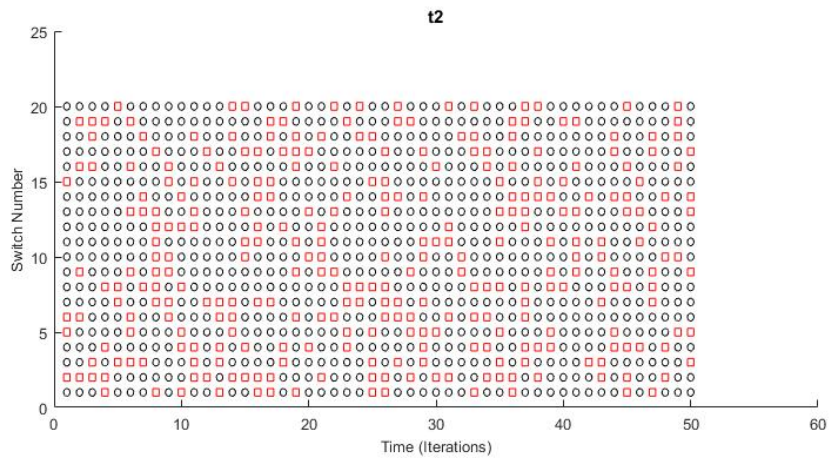


Figure 5.25: Control plane states for DO with t_2 in random conditions

As for the hi functions, we used the linear model as follows:

$$h_1(\mathbf{d}) = \mathbf{d} \tag{5.41}$$

$$h_2(\mathbf{t}) = 1 - \mathbf{t} \tag{5.42}$$

$$h_3(\mathbf{p}) = 1 - \mathbf{p} \tag{5.43}$$

$$h_4(\mathbf{n}) = \mathbf{n} \tag{5.44}$$

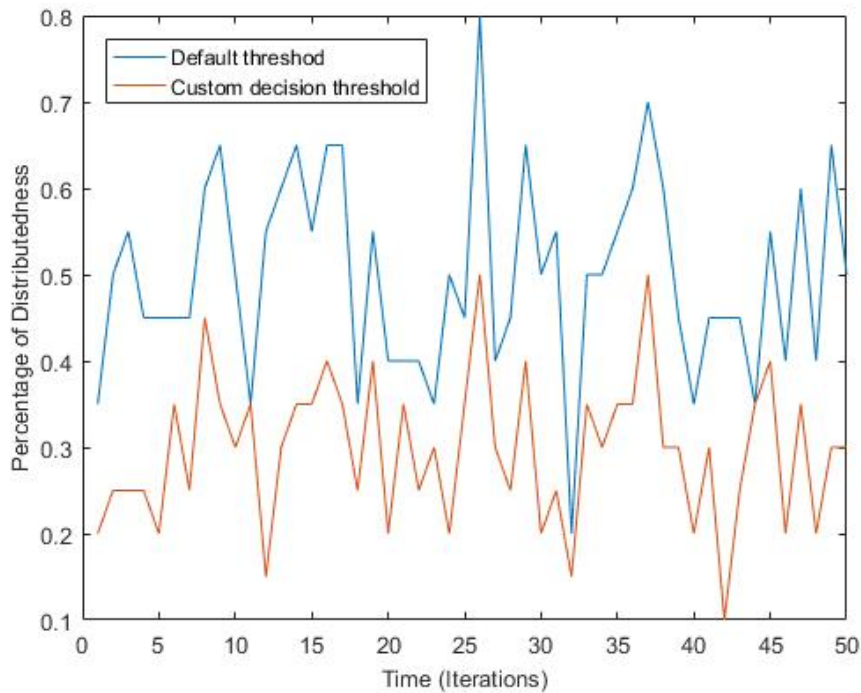


Figure 5.26: Percentage of HS's having distributed control wrt. time

Figure 5.24 and 5.25 show the control states of the different HS's through time, where a black circle represents a centralized HS, and a red square represents a distributed HS. When comparing both figures, we notice that threshold t_1 (Figure 5.24) results in a network that has approximately same number of red and black HS's, whereas thresholds t_2 , which favors the centralized state, results in a higher number of black HS's (Figure 5.25), as expected. This observation is verified in Figure 5.26, which shows that the custom threshold t_2 (in red) results in a lower percentage of distributed HS's in each iteration when compared to the default threshold t_1 (in blue). Figure 5.27 shows the total percentage of distributed HS's over all iterations for the two thresholds. We notice that for the default threshold the total percentage is 50%, which means that approximately half of HS's were distributed. For threshold t_2 , this percentage drops to around 30%.

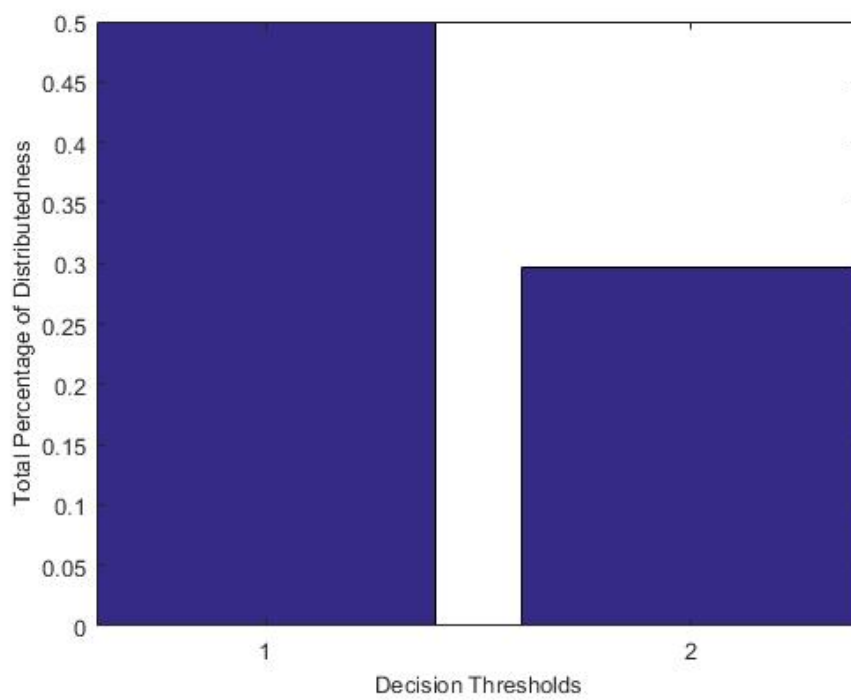


Figure 5.27: Total percentage of HS's having distributed control over the total time period

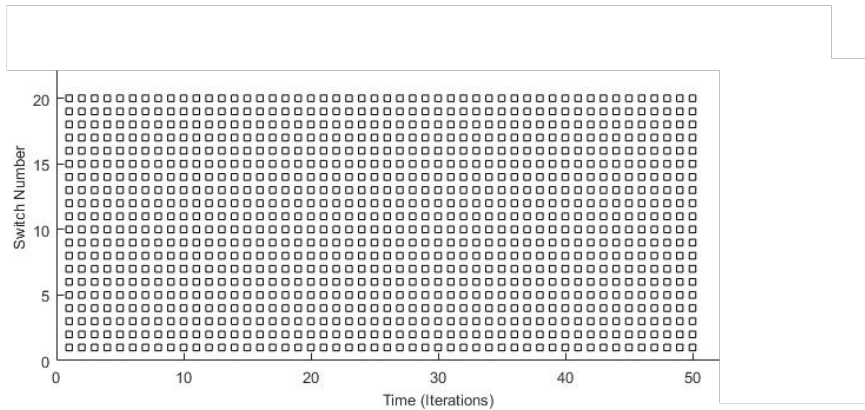


Figure 5.28: Control plane states for fully centralized network conditions

Boundary Study: Fully Centralized

Conditions for a fully centralized network are:

$$\mathbf{d} = 0 \rightarrow \mathbf{h1}(\mathbf{d}) = 0$$

$$\mathbf{t} = 1 \rightarrow \mathbf{h2}(\mathbf{t}) = 0$$

$$\mathbf{p} = 1 \rightarrow \mathbf{h3}(\mathbf{p}) = 0$$

$$\mathbf{n} = 0 \rightarrow \mathbf{h4}(\mathbf{d}) = 0$$

Under these conditions, the closed form solution becomes:

$$\mathbf{s} = \frac{0 + 0 + 0 + 0}{\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4} = 0$$

As expected, these conditions result in a network that is fully centralized at all times (black HS's in Figure 5.28). This is also verified in Figures 5.29 and 5.30 which show a 0 percentage of distributed HS's.

Boundary Study: Fully Distributed

Conditions for a fully distributed network are:

$$\mathbf{d} = 1 \rightarrow \mathbf{h1}(\mathbf{d}) = 1$$

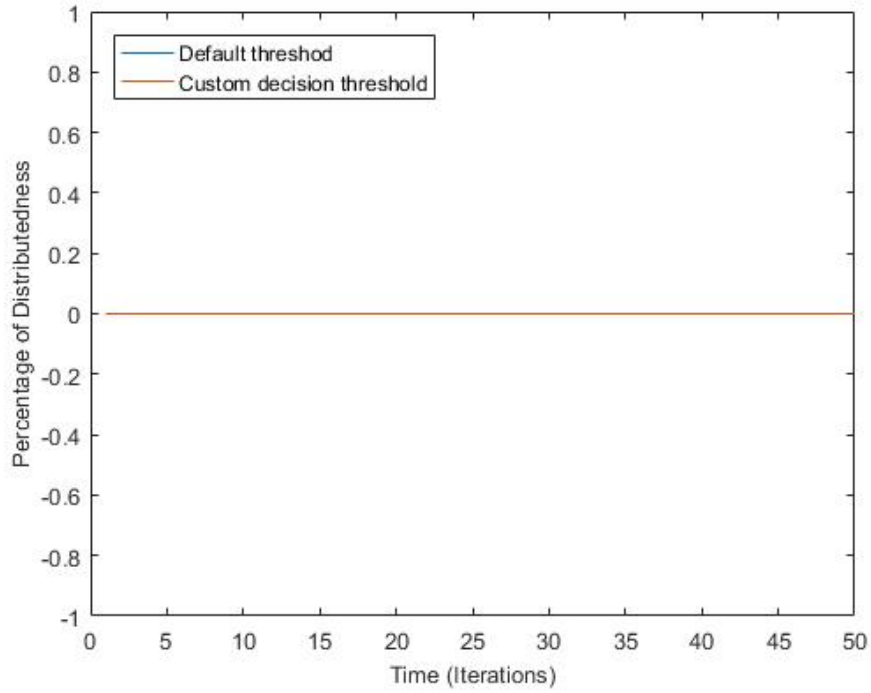


Figure 5.29: Percentage of HS's having distributed control wrt. time

$$t = 0 \rightarrow h_2(t) = 1$$

$$p = 0 \rightarrow h_3(p) = 1$$

$$n = 1 \rightarrow h_4(d) = 1$$

Under these conditions, the closed form solution becomes:

$$s = \frac{\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4}{\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4} = 1$$

Under fully distributed network conditions, the optimization system results in a fully distributed network at all times, as indicated by the red HS's in Figure 5.31. Figures 5.32 and 5.33 also verify these results.

The results above prove that our optimization model respects the boundary conditions for s .

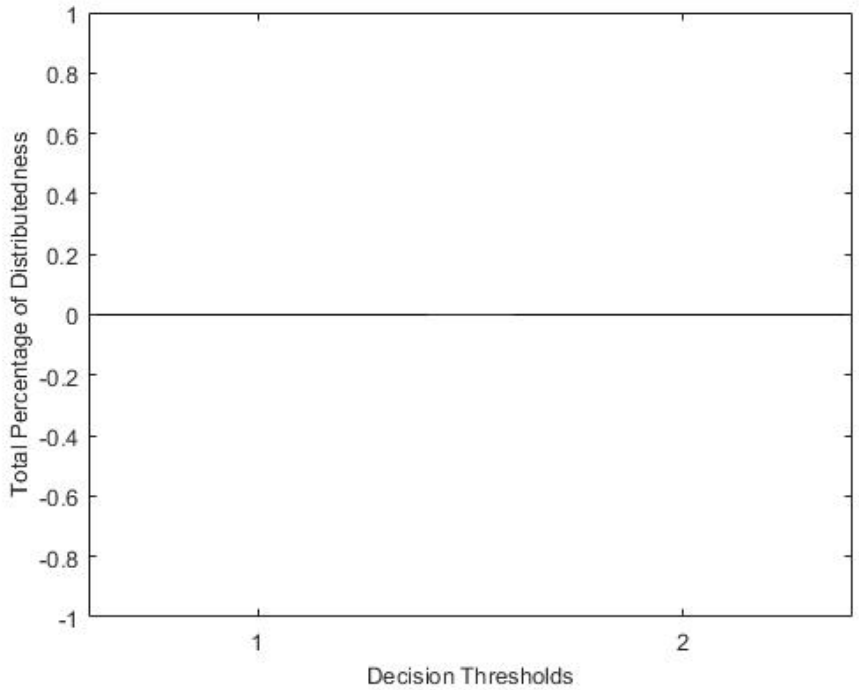


Figure 5.30: Total percentage of HS's having distributed control over the total time period

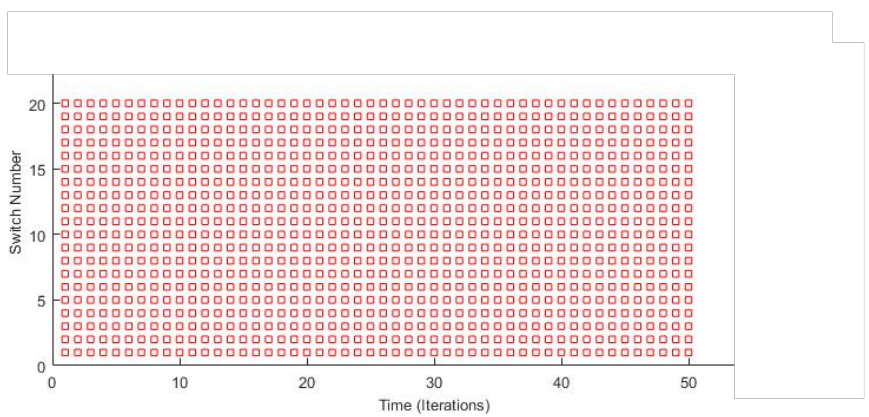


Figure 5.31: Control plane states for fully distributed network conditions

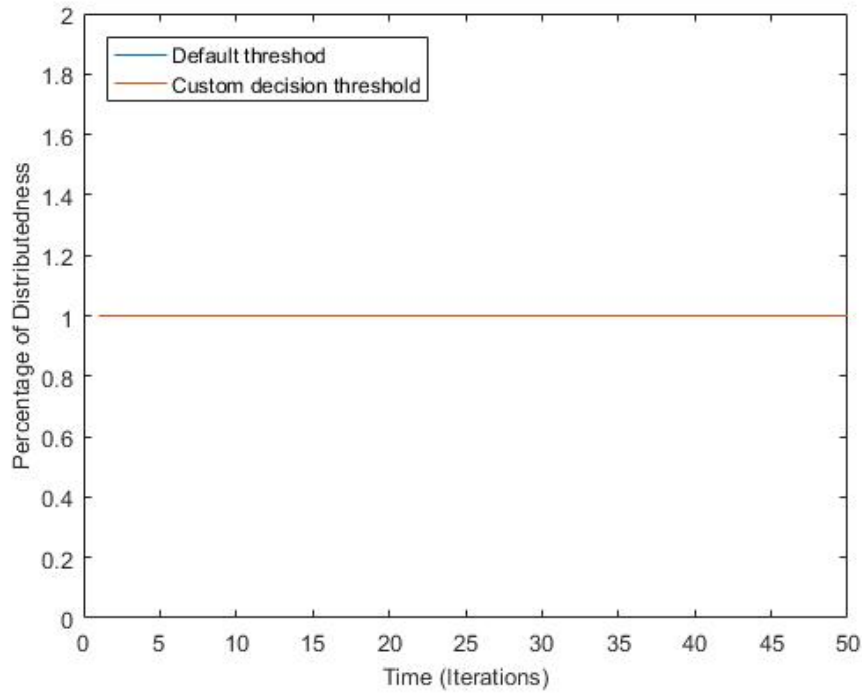


Figure 5.32: Percentage of HS's having distributed control wrt. time

Incremental Network Conditions

We also tested the DO algorithm on incremental network conditions; at the first iteration, the network conditions favor centralization. They are incremented at each iteration towards decentralization to reach values that favor decentralization at the 100th iteration. The system was tested on a network of 100 switches. Results are shown in Figures 5.34 and 5.35, for thresholds t_1 and t_2 respectively.

When comparing the two figures, we notice the effect of the threshold on the decision. In Figure 5.34, as t_1 provides 50% chance for both states, we notice that the transition from centralization to decentralization is around the 50th iteration. However, in Figure 5.35, this transition happens later as expected (around the 80th iteration), since t_2 favors the centralized state. Figures 5.34 and 5.35 also show that the DO model is validated at the boundaries: at iteration 0,

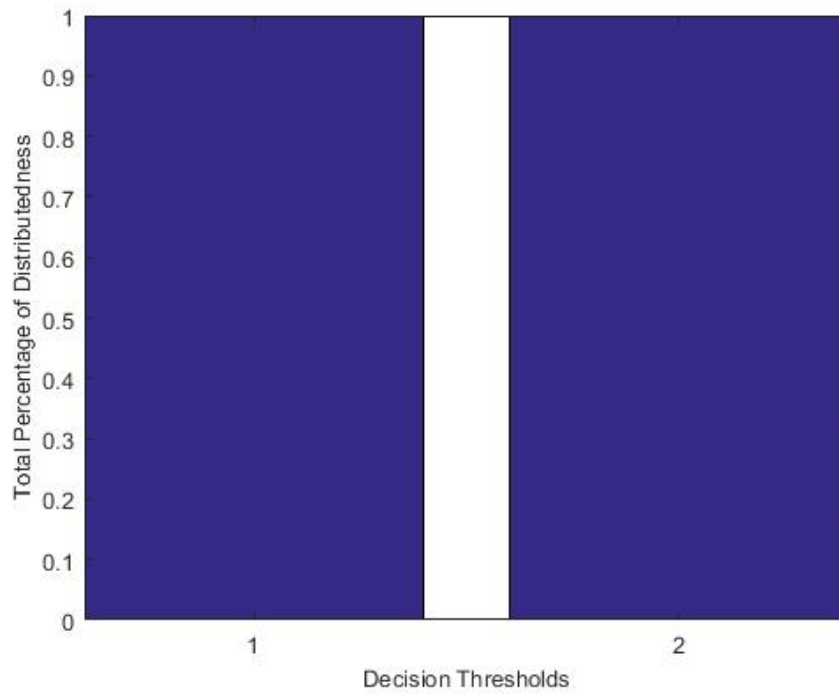


Figure 5.33: Total percentage of HS's having distributed control over the total time period

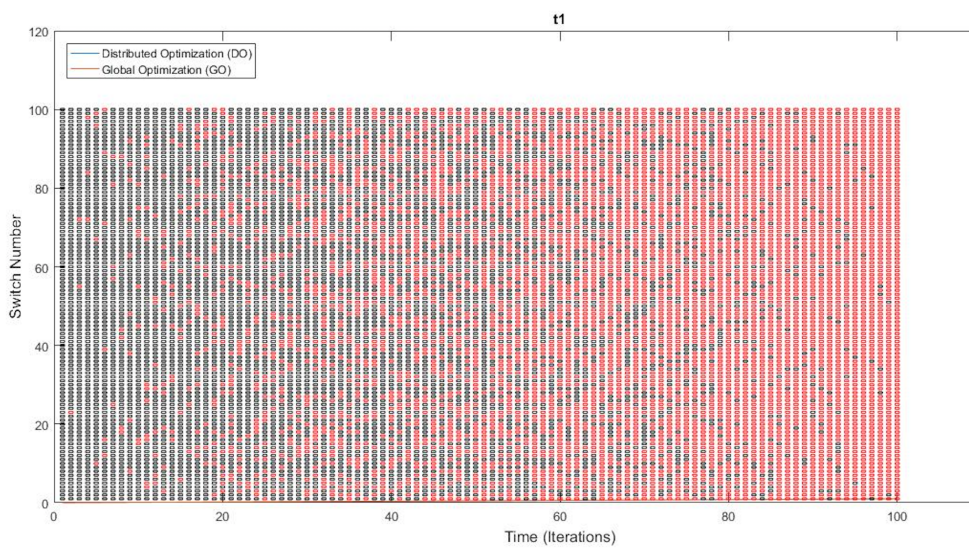


Figure 5.34: Control plane states for DO with t_1 in incremental conditions

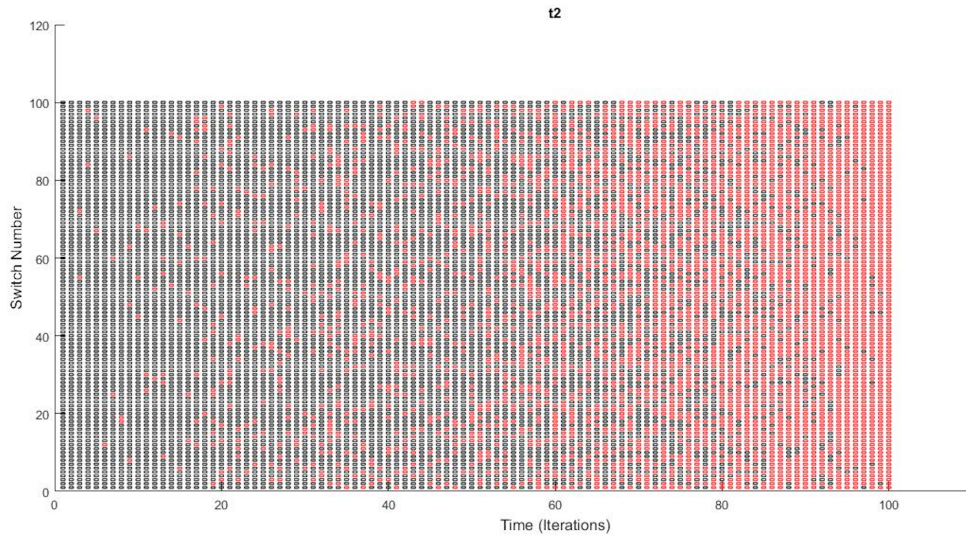


Figure 5.35: Control plane states for DO with t_2 in incremental conditions

when conditions are for a fully centralized network, we notice that all nodes are centralized (all black points). Alternatively, at the 100th iterations, all nodes are decentralized (red points).

Figure 5.36 shows the total % of decentralization in the network. We notice that the system using t_2 (red curve) has a slower increase towards the full decentralized state than that of the blue curve (system with t_1) . Results of the incremental network conditions are in line with the expected behavior of the DO.

5.6.2 GO Testing

We also tested the global optimization, first as a standalone solution where parameter d is used to decide on which switches should be centralized (design alternative 5.5.3), and second as a bi-level optimization where the results of DO are fed into GO for the final decision making. We used Matlab to simulate a network of 100 nodes, with varying network conditions as per equations 5.37 to 5.40. Figure 5.38 shows the state of the 100 switches as network conditions evolve from

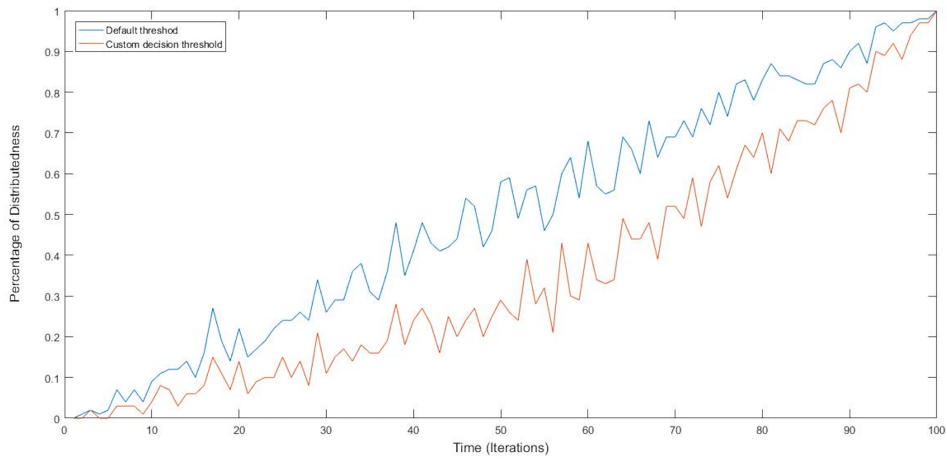


Figure 5.36: Percentage of distributed control vs. time

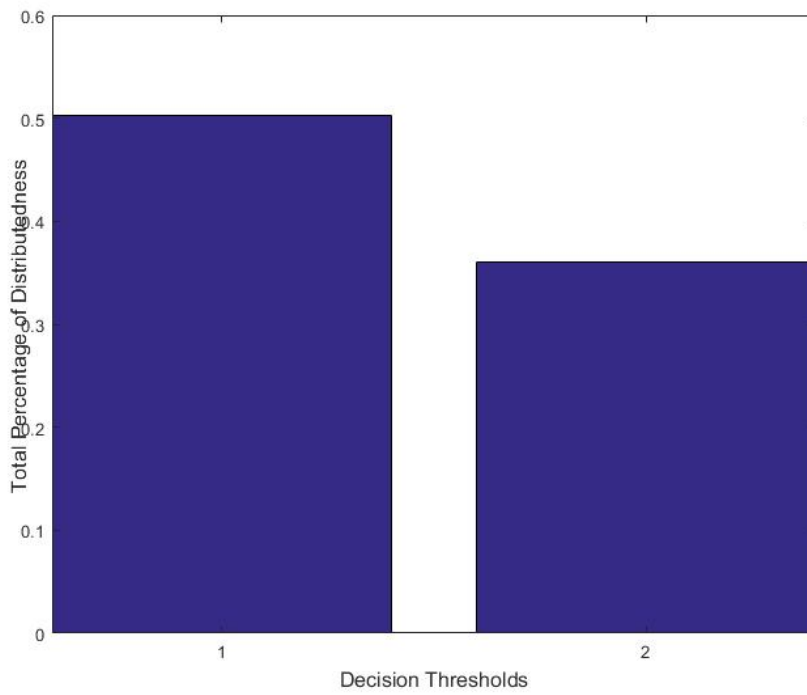


Figure 5.37: Total percentage of distributed control

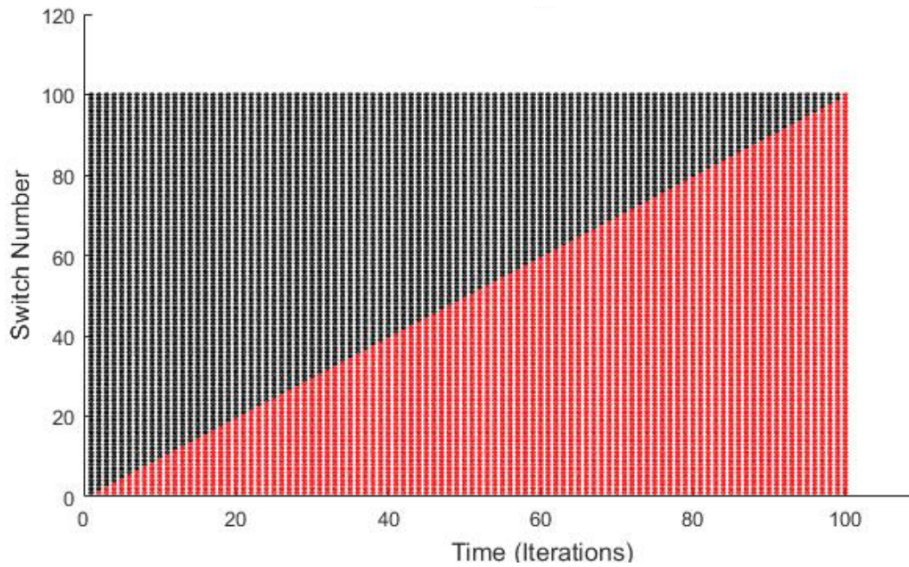


Figure 5.38: Control plane states for standalone GO in incremental conditions favoring centralization to favoring decentralization, under the standalone GO. At the boundaries, we see that the model converges to full centralization when network parameters favor this state (iteration 0), and becomes completely decentralized when network parameters favor decentralization (iteration 100), which corresponds to the expected behavior of the system.

Figure 5.39 compares the total % of decentralized switches for DO and standalone GO. The response of the standalone GO is a curve that increases linearly from 0 to 100%, whereas the response of the DO is less stable, and fluctuates around the straight line, with however the same trend. This shows that the GO is more stable than the DO and is less sensitive to small variations of the network conditions.

Finally, Figure 5.40 shows the evolution of the control states in the network, again under incremental conditions as is the case for Figure 5.38, but for the bilevel GO-DO system. Similarly to the standalone GO system (Figure 5.38), this system also obeys boundary conditions. The difference in the color pattern

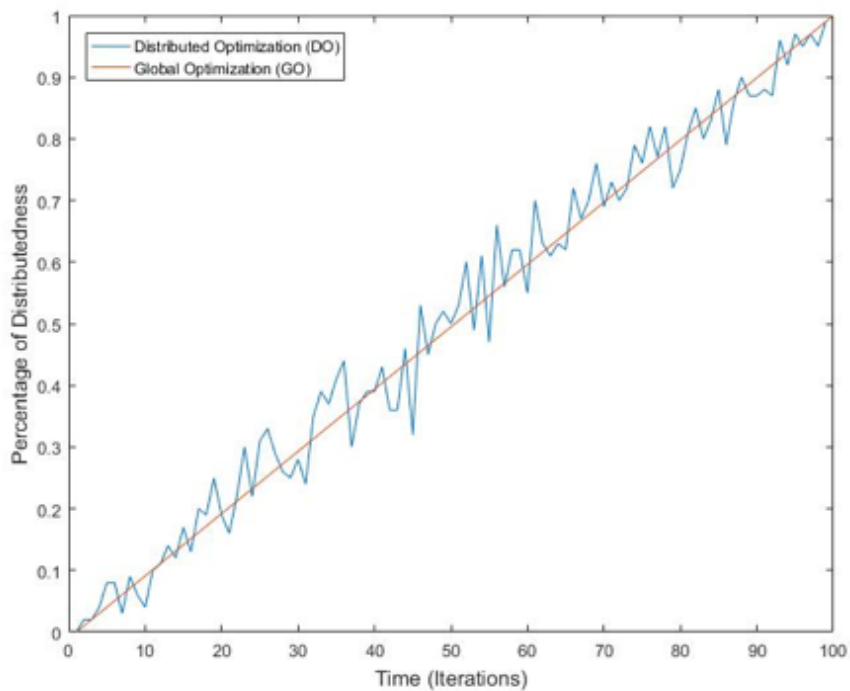


Figure 5.39: Percentage of distributed control vs. time

between the two figures is the decision variable used to decide on switches: in Figure 5.38, parameter d was the deciding variable, and given that d was being incremented linearly starting from delay on switch 1 to 100, we were able to see the symmetric pattern between the black and red dots as the network became more decentralized (i.e. as d increased). In Figure 5.40, the decision is based on the result of the underlying DO algorithm, which is by itself controlled by all 4 parameters and not just parameter d .

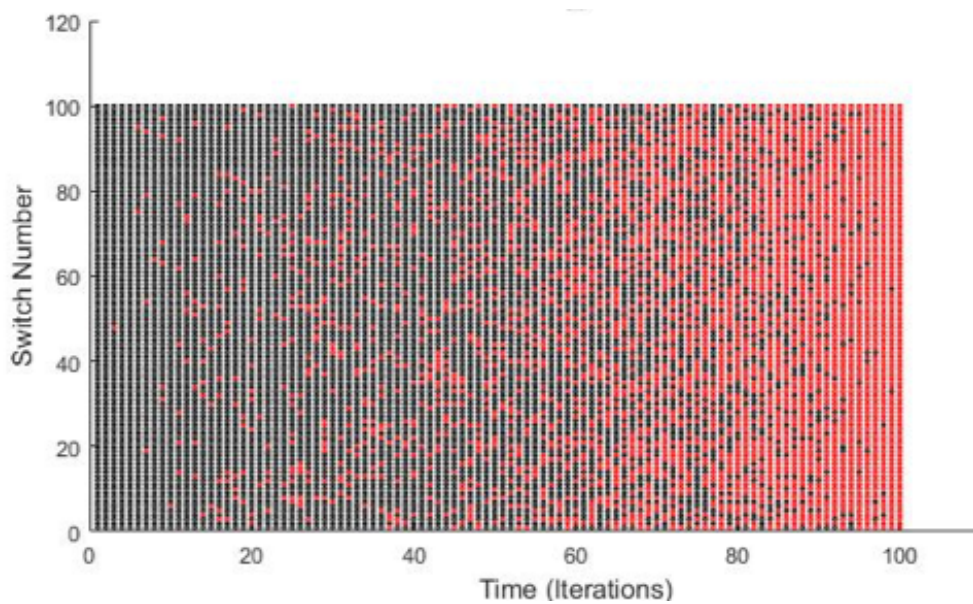


Figure 5.40: Control plane states for our proposed optimization in incremental conditions

Chapter 6

Implementation and Results of Final Complete System

6.1 Implementation Details

Throughout this dissertation, different components of the network were designed and tested separately. The optimization systems were simulated with MATLAB, and tests were performed to verify that the behavior of the system is in line with the control rules. Performance testing of SDN and OSPF was completed using python and the MiniNEXT simulator. The final implementation presented in this section combines all the subsystems that constitute the work. The different modules that come into play are illustrated in 6.1.

As shown in the diagram in 6.1, the implementation system includes 6 modules:

1. The hybrid switch (the HS block)
2. The SDN controller block

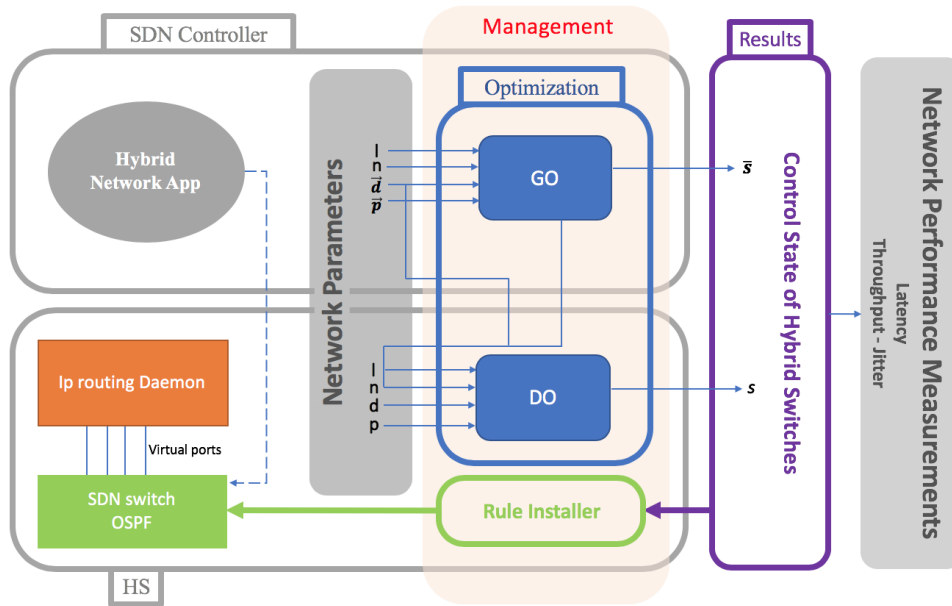


Figure 6.1: Implementation design diagram

3. The HS state control (the Rule Installer block)
4. The monitoring system (the Network Parameters)
5. The optimization system (the Optimization block)
6. And the network performance measurements (The Measured Performance)

The following details the implementation of each block.

6.1.1 The Hybrid Switch (HS):

Designing and implementing the HS is the main task, as it constitutes the building block of the hybrid network. We previously presented OSHI [49] as a testbed for hybrid networks; the OSHI node is a hybrid node that supports both SDN and IP protocols and runs them simultaneously. It is a linux container which implements OVS and the Quagga suite with OSPF as the enabled IP routing protocol. The coexistence of the control planes is ensured by exploiting the concepts of multiple

flow tables and virtual ports. OSHI implements traffic segregation, and uses VLL and PW techniques to process classes of traffic with SDN and other classes with OSPF. In the OSHI network, the ingress OSHI nodes tag the traffic to separate between SDN flows and OSPF flows, and the egress nodes remove the tags (if present) before delivering traffic to destination. On the OSHI core nodes, the simulator installs rules to forward packets received on the physical ports to the corresponding virtual ports, where they will be delivered to OSPF for processing, and rules to forwards tagged traffic to table 1 which contains the SDN controller flows.

The authors in [149] implement a similar scenario, where the network shifts from SDN to OSPF based on the packet loss ratio of the control channel. In their work, the network is homogeneous, i.e. the whole network is either SDN or OSPF, unlike our hybrid network where the switches don't have to be in the same control states. The authors use the technique implemented by OSHI to add/pop VLAN tags to/off packets. On the switches, two forwarding tables exist. Rules installed by the SDN controller either send received packets to the virtual ports where the OSPF daemon is running, or to table 1 where the SDN forwarding flows are installed. Consequently, the control is done at the data plane level, where tagging or un-tagging packets is the mechanism to force packets to be handled by SDN or OSPF.

The HS Design

Although the OSHI nodes are hybrid nodes, their functionalities does not correspond exactly to our HS design. We did however use the simulator with some changes to the nodes' behavior, to adapt them to the behavior of the HS. There are two main implementation differences between our design and OSHI. We can-

celed the VLL deployment and the VLAN tagging, since the HS is either fully centralized or fully decentralized, and thus traffic segregation is not adopted; instead, all traffic is treated in the same manner depending on the control state of the HS. Also, we use the same generic HS based on a generic OSHI node, instead of defining Ingress/Egress and Core OSHI nodes like it is the case in the OSHI simulator. To change the HS control state, rules are added to - or deleted from the switch via the rule installer module. Also, we take advantage of the multi-table functionality in the OVS to group the rules that correspond to the distributed control plane in table 0, and those of the centralized control plane (SDN controller rules) in table 1.

6.1.2 The HS State Control System – the Rule Installer Block:

In addition to the double routing protocols functionality, the HS houses a rule installer module, in charge of enforcing the control state of the HS. It forces the HS to switch from centralized to distributed control or vice-versa given the results of the DO or the GO, whichever is running.

The python code sits in the rule installer in the HS, and directly injects the corresponding OVS rules to this latter. These rules dictate the behavior of the HS with respect to received packets as follows.

OSPF Mode

As per the HS design, the rules in table 0 of the OVS switch map physical ports to virtual ports; i.e. when in this mode, all packets received will be sent to the

```

cookie=0x0, duration=4380.092s, table=0, n_packets=6, n_bytes=468, priority=300
,in_port=8 actions=output:7
cookie=0x0, duration=4380.053s, table=0, n_packets=4, n_bytes=300, priority=300
,in_port=7 actions=output:8
cookie=0x0, duration=4380.007s, table=0, n_packets=19811, n_bytes=1511095, prio
rity=300,in_port=6 actions=output:5
cookie=0x0, duration=4379.973s, table=0, n_packets=19588, n_bytes=2777004, prio
rity=300,in_port=5 actions=output:6
cookie=0x0, duration=4379.947s, table=0, n_packets=927, n_bytes=72566, priority
=300,in_port=4 actions=output:3
cookie=0x0, duration=4379.915s, table=0, n_packets=46, n_bytes=3940, priority=3
00,in_port=3 actions=output:4
cookie=0x0, duration=4379.871s, table=0, n_packets=26, n_bytes=3556, priority=3
00,in_port=2 actions=output:1
cookie=0x0, duration=4379.823s, table=0, n_packets=15, n_bytes=2878, priority=3
00,in_port=1 actions=output:2

```

Figure 6.2: Table 0 rules for OSPF mode

appropriate virtual twin port. An example of the set of these rules is shown in Figure 6.2. These rules are from an HS with 4 physical ports, and 4 virtual ports accordingly, and we can observe the following port mapping:

1. in_port=8 \rightarrow output:7 (first rule) and
in_port=7 \rightarrow output:8 (second rule)
2. in_port=6 \rightarrow output:5 (third rule) and
in_port=5 \rightarrow output:6 (fourth rule)
3. in_port=4 \rightarrow output:3 (fifth rule) and
in_port=3 \rightarrow output:4 (sixth rule)
4. in_port=2 \rightarrow output:1 (seventh rule) and
in_port=1 \rightarrow output:2 (eighth rule)

SDN Mode

To make the HS rely on the centralized control plane, we force all received packets to be sent to table 1 of the OVS switch. In this table, we find the normal

```

OFPST_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x0, duration=16.400s, table=0, n_packets=6, n_bytes=468, ip actions=got
  o_table;1
  cookie=0x0, duration=16.383s, table=1, n_packets=0, n_bytes=0, ip,in_port=4 act
  ions=output;5
  cookie=0x0, duration=16.363s, table=1, n_packets=0, n_bytes=0, ip,in_port=5 act
  ions=output;4

```

Figure 6.3: Table 1 rules for SDN mode

forwarding rules that the controller injects to route traffic. Example of these rules are shown in Figure 6.3. The first rule is in table 0; it matches all IP packets, and sends them to table 1. The next two flows are in table 1 (as shown in picture), and they do normal packet forwarding from one physical port to the other as per the SDN controller’s installed instructions.

6.1.3 The Monitoring System:

Normally, the monitoring system measures the system parameters and invokes the optimization system at each T_o . In our experiment, we actually set these system variables, to observe their effect on the network performance in a controlled manner. We use different tools to enforce the values of the 4 systems parameters to recreate the effect of changing network conditions.

1. Delay to controller: the netem [150] tool was used to set the delay on the control channel between the controller and the SDN switches. This tool allows the user to specify the latency on a given interface in the network.
2. Rate of Unknown packets: we also use the netem tool with the corrupt, duplicate, and reorder options to inject random traffic in the network.
3. Network size: The network size can be controlled by adding and deleting HS’s from the network. This parameter can be read via a REST API from the SDN controller [151, 152].

4. Topology change rate: We developed a tool that takes the topology change rate as input and provokes a series of Mininet's "link up/down" commands for each T_0 throughout the simulation depending on the specified rate.

6.1.4 The Optimization System/Decision Making:

The optimization system implements the closed loop solutions for the Global Optimization (GO), the Decentralized Optimization (DO) elaborated in section 5.5. The DO algorithms can return two types of results based on the type of threshold chosen (a fair threshold and a threshold that favors centralization, refer to section 5.6.1). As for the GO, the algorithms implements two decision making alternatives; decision making based on the result of the DO, or based on the values of the switch/controller delays. The optimization system with all its variations was initially developed in Matlab, and then translated to python. As shown in Figure 6.1, the GO sits on the controller, which is in charge of running the global optimization, whereas the DO is implemented on each Hybrid Switch, which are themselves responsible for running the DO algorithm. We created 2 optimization functions, namely GOfunction and DOfunction, and an optimization script that calls the functions depending on the chosen design alternative, passes the network parameter values to the function, and receives the optimal states of the HS's in the network.

6.1.5 SDN Controller

We chose Ryu SDN controller since it is fully written in python [55], which makes the implementation system more homogeneous, given that all system's components are written in python as well. Based on the "simple.switch.13.py" Ryu

application, we create the "Hybrid Network App" to support the hybrid network that we designed. The Hybrid Network App implements multiple Openflow tables on the HS's, and downloads the SDN controller flows to the HS's tables 1 instead of table 0, as per the functionalities described in section 6.1.1. GO is implemented in python at the controller's level. The GO module sits on top of the Hybrid Network App, it periodically reads the parameters of the network, and recomputes the GO algorithm, depending on the optimisation design alternative that is adopted (refer to 5.5.3).

6.1.6 Experiment Overview

After developing all items above, all subsystems were grouped on the testing machine. The specs are shown below:

- Virtual box version 6.0.14
- OS - Ubuntu Linux 13.04
- OSHI v7
- Mininet v2.1.0

In order to bound the experiment, we set the maximum allowed values for the network parameters to be as follows:

- The maximum delay that can be observed on the link to SDN controller, d_{\max} is assumed to be 100ms. This maximum value represents the loss of communication to the controller.
- The maximum allowed rate of topology change t_{\max} is assumed to be 1 change/ s^{-1} , with $T_0 = 1$ minute.

- The maximum allowed rate of unknown traffic is set to be 40%, because the experimental study that we conducted showed that the OSPF network broke at 40% corruption rate
- The maximum network size is set based on the capacity of the chosen controller. For these experiments, we set the maximum size to 100.

To measure network performance, we perform measurements using the ping command and iperf [153] between the two hosts h1 and h2. Ping measures latency, whereas Iperf tests the network for its bandwidth, throughput, and jitter. We ran each experiment 20 times for a period of $10 \times T_0$ minutes for each run. The scenarios and the corresponding results are reported below.

6.2 Testing and Results

The testing process workflow is shown in Figure 6.4. For each scenario defined below, the testing process starts by starting the topology with the OSHI simulator, using however the modified OSHI nodes for the HS's (as per section 6.1.1), then starting the Ryu controller and running the Hybrid Network App. Next at each T_0 , a python script sets the values of the system variables, calculates the optimization results, configures the control states of the switches, and measures the resulting network performance with iperf and ping commands.

This testing is repeated for the two thresholds of the DO, the two decision-making methods of the GO, an all SDN network, and an all OSPF network, in the aim of comparing the performance of the different network designs.

The reference network architecture is shown in figure 6.5.

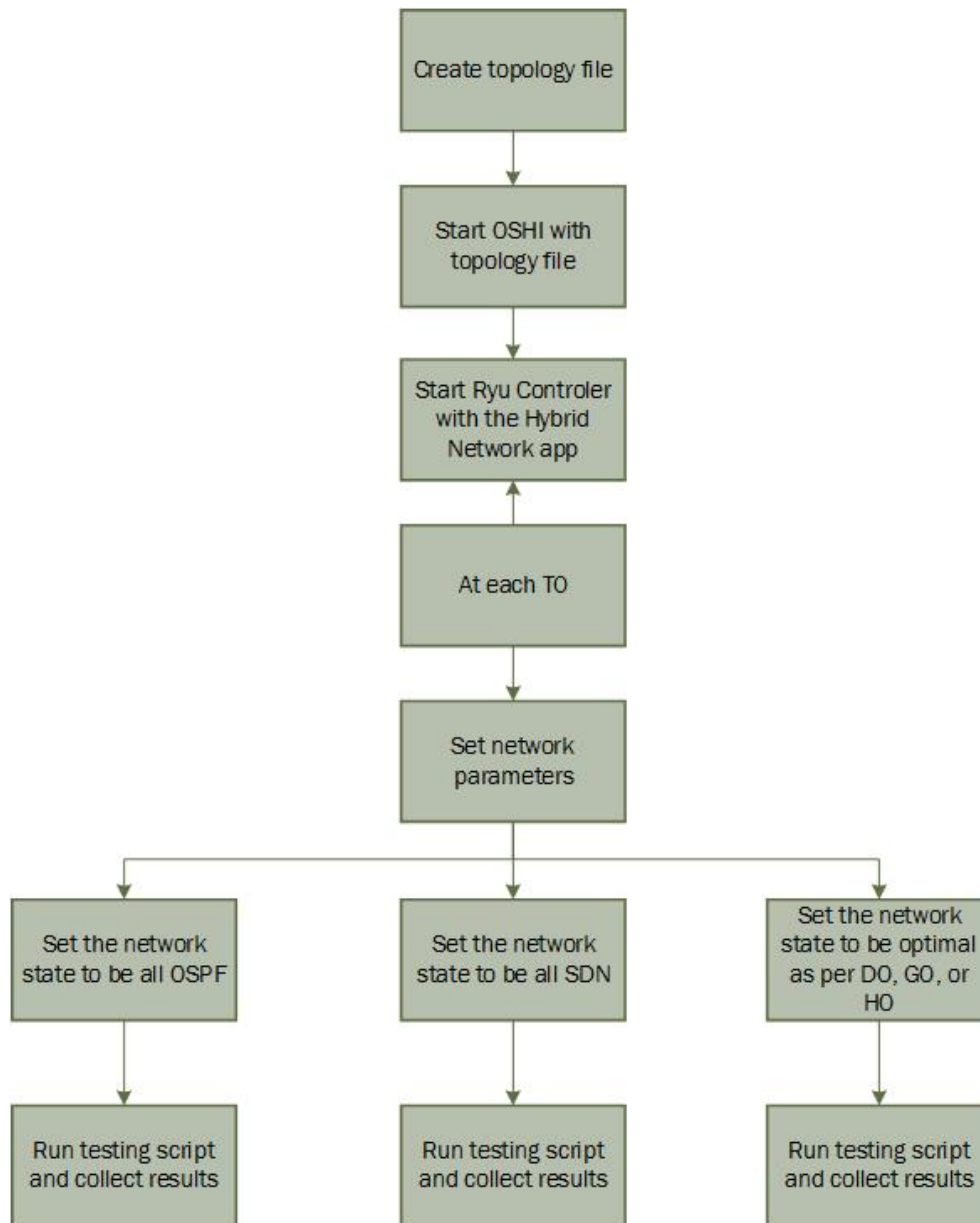


Figure 6.4: Testing workflow

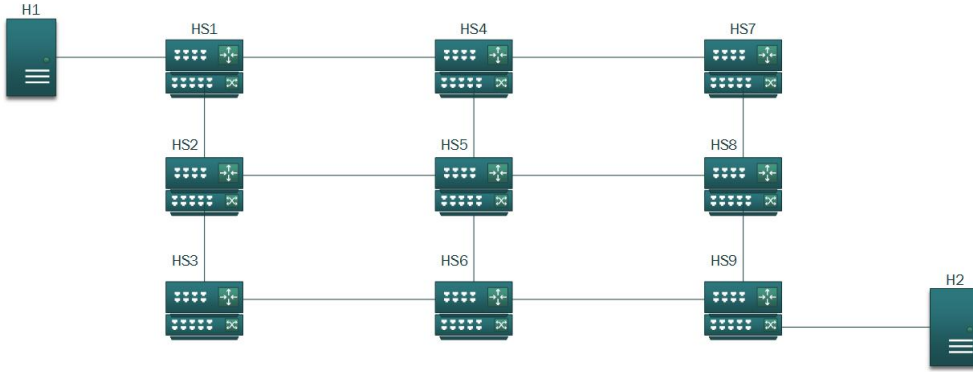


Figure 6.5: Network architecture

Network value	Normalized value
$n = 10$	$n = 0.1$
$t = 1T_0^{-1}$	$t = 0.17$
$p = 0 \%$	$p = 0$
$d = [0 - 100] \text{ ms}$	$d = [0 - 1]$

Table 6.1: Parameters for scenario 1

6.2.1 Scenario 1: Effect of Communication Delay with Controller

In the first scenario, we set the network size to be 50% of the maximum network size which corresponds to 9 nodes and 7 links. The rest of the parameters are set to stable topology and stable traffic. As for the delay to controller, we sweep over values of the delay from 0 to maximum delay, to study the effect of the delay to controller on the result of the 4 optimization schemes, and consequently the network performance. For the purpose of the experiment, and in order to emphasise the effect of the delay, we set the coefficient α_1 of the delay parameter to 10.

The parameters values are reported in table 6.1.

Results are shown below. The control states of the HS's throughout the iterations are compiled in Figures 6.6 to 6.9.

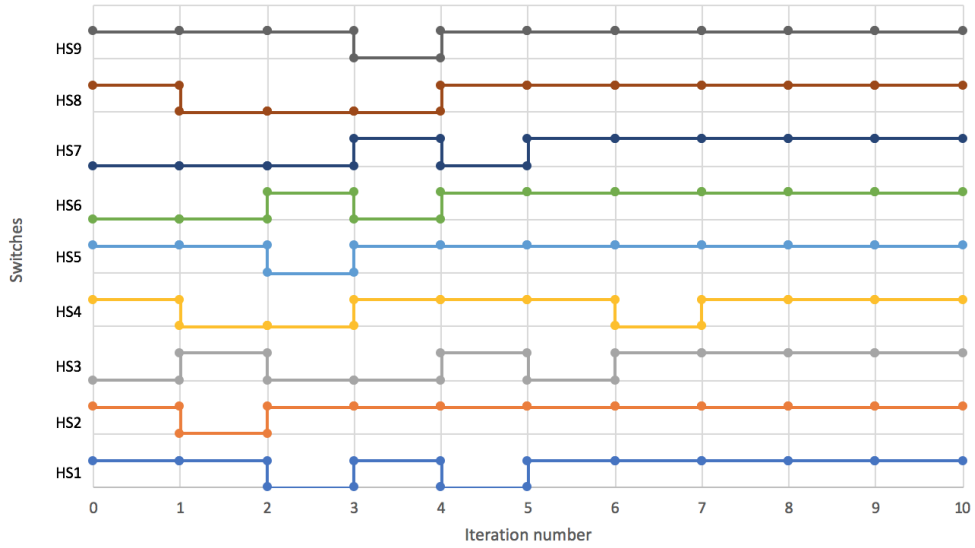


Figure 6.6: HS states for DO with threshold t_1

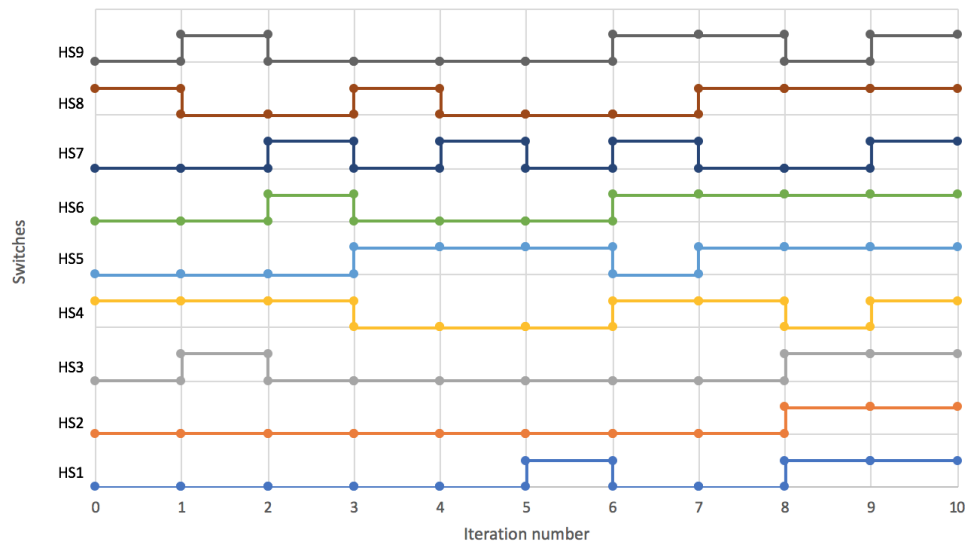


Figure 6.7: HS states for DO with threshold t_2

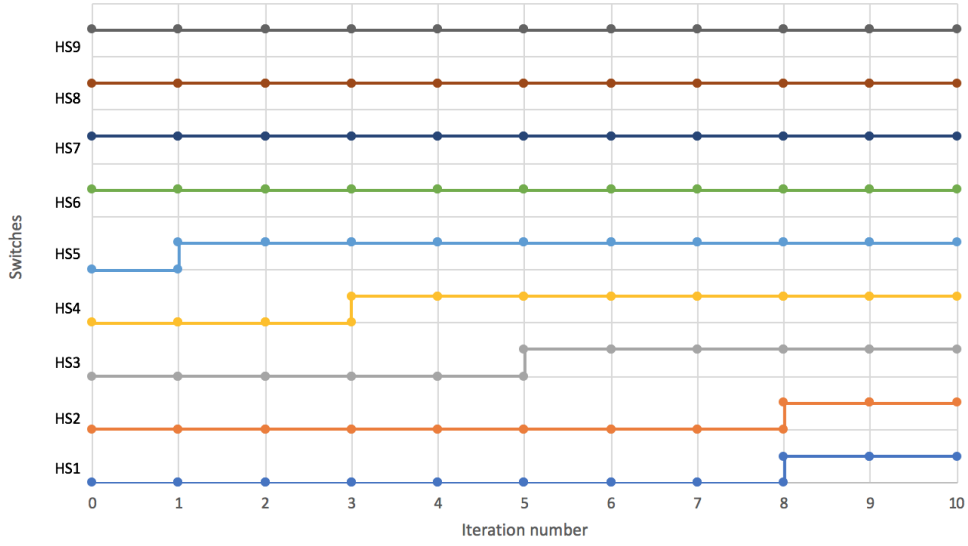


Figure 6.8: HS states for GO with decision based on results of DO

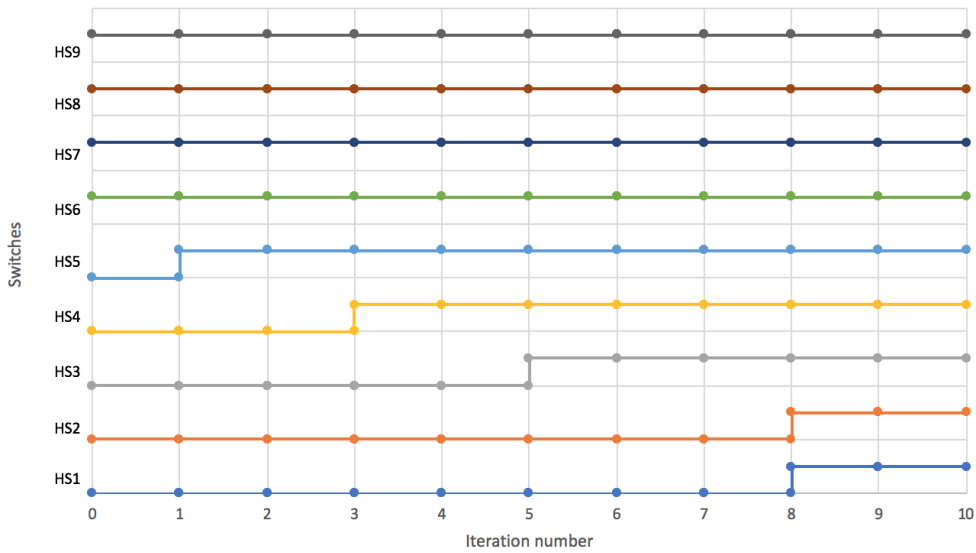


Figure 6.9: HS states for GO with decision based on delay values

Figure 6.10 shows the latency for the 4 network designs as the delay to controller increases.

Results show that in the chosen scenario, the optimizations systems favor decentralized networks, which is expected, because the system parameters in general favor decentralization, especially that an increasing d hurts a centralized system. It is worth noting that the latency in SDN and OSPF is comparable, this is due to the fact that when the HS's have the flow rules installed already, then the delay to controller won't affect performance much, unless an event happens that pushes HS to ask for controller's help, like a port failure for example. We simulate this behavior by performing one topology change for each T_0 and for each value of d to force the HS with centralized control plane to solicit the controller. We notice that DO_{t1} gives the best performance among the 4 optimization designs. In fact, when compared to DO_{t2} , this latter favors centralization, and thus the complete shift to distributed network takes place later than it is the case in DO_{t1} . However, a network operator would by choice decide to use a threshold that favors centralization, if for example network services that require SDN are deployed. In this case, the operator would pay the price of slightly increased performance compared to OSPF, but yet increased performance compared to pure SDN, which suffers from bad switch/controller links. We notice that when the delay increases to almost reach the maximum delay allowed on the network, all optimization converge to an all OSPF network. This is also expected, because if the delay to controller reaches the point considered as a loss of communication to controller, then trying to communicate to a non-responsive controller is pointless.

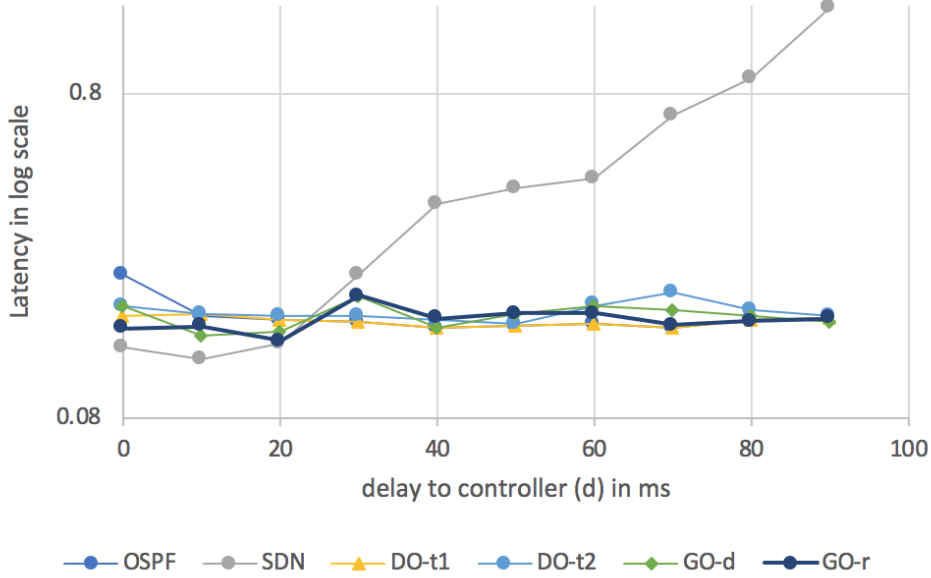


Figure 6.10: Latency with respect to delay for the different network designs in semi-log scale

Network value	Normalized value
$n = 10$	$n = 0.1$
$t = [0, 60]$ changes/ T_0	$t = [0, 1]$ changes/sec
$p = 1\%$	$p = 0.01$
$d = 30$ ms	$d = 0.3$

Table 6.2: Parameters for scenario 2

6.2.2 Scenario 2: Effect of Topology Change Rate

In Scenario 2, we vary the stability of the topology, by applying a series of link down/up commands in Mininet, within a period of $T_0 = 60$ sec. The rate of topology changes varies between 0 and 1, which corresponds to 0 and 60 changes per T_0 . The rest of the parameters are set to stable topology, stable traffic, and small delay to controller.

The parameters values are shown in table 6.2.

Results are shown below. Figures 6.11 to 6.14 show the states of the HS's for each iteration. We notice that the 4 optimization systems converge to central-

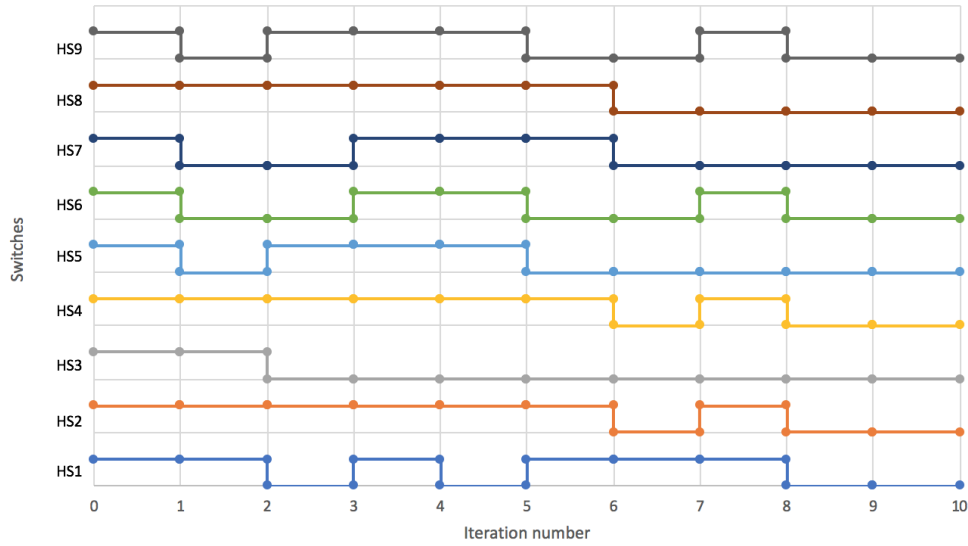


Figure 6.11: HS states for DO with threshold t_1

sation and full centralization, as per our control rule.

Figure 6.15 shows the latency for the 4 optimization designs as the rate of topology change increases. We notice that the decentralized system performs well up to a certain rate where it's performance break. This is probably due to the increase of control messages rippled throughout the network links. Performance of SDN is also affected by the increased topology change rate, however it is also affected by the delay of it's control plane, which we set to 30ms to emphasise the behavior of the optimized networks. Both DO systems start as OSPF networks. This is expected because for a topology change rate of 0, a higher delay to controller pushes the systems to be more decentralized. As the rate of topology change increases, the two DO become more centralized. The switch to the centralized states is faster with DO_{t_2} than with DO_{t_1} .

The latency results in Figure 6.15 show that overall, the optimized systems perform better than OSPF or SDN alone, especially at the boundary conditions. The DO systems perform better than the GO systems, but these latter do perform

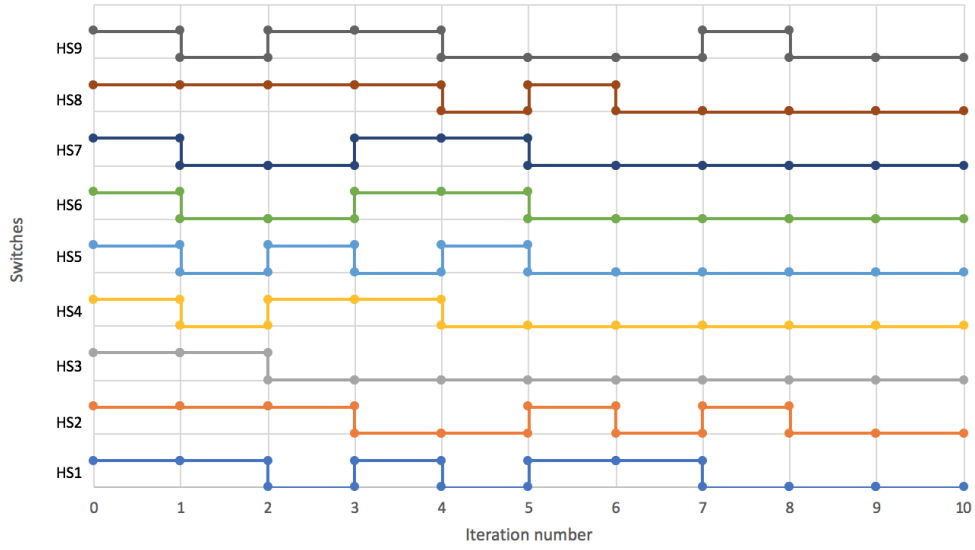


Figure 6.12: HS states for DO with threshold t_2

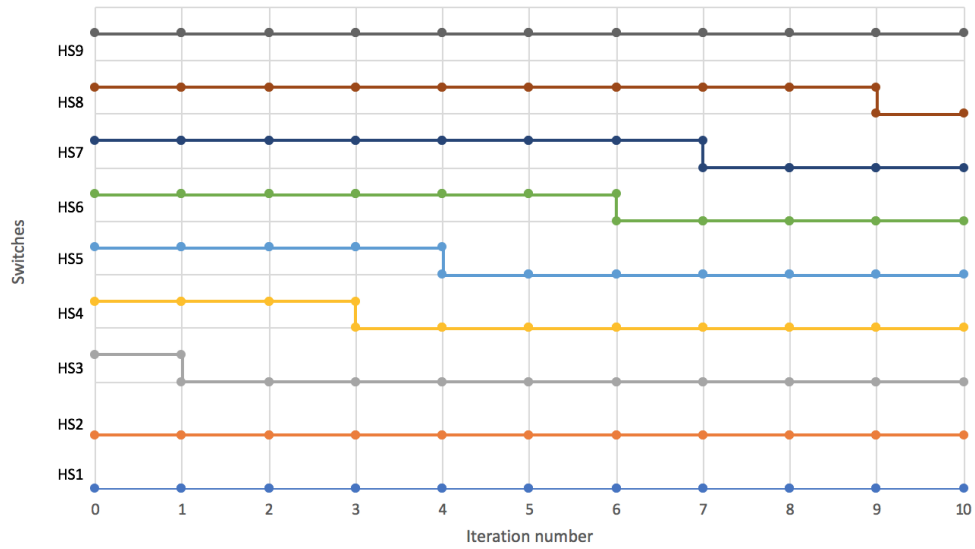


Figure 6.13: HS states for GO with decision based on results of DO

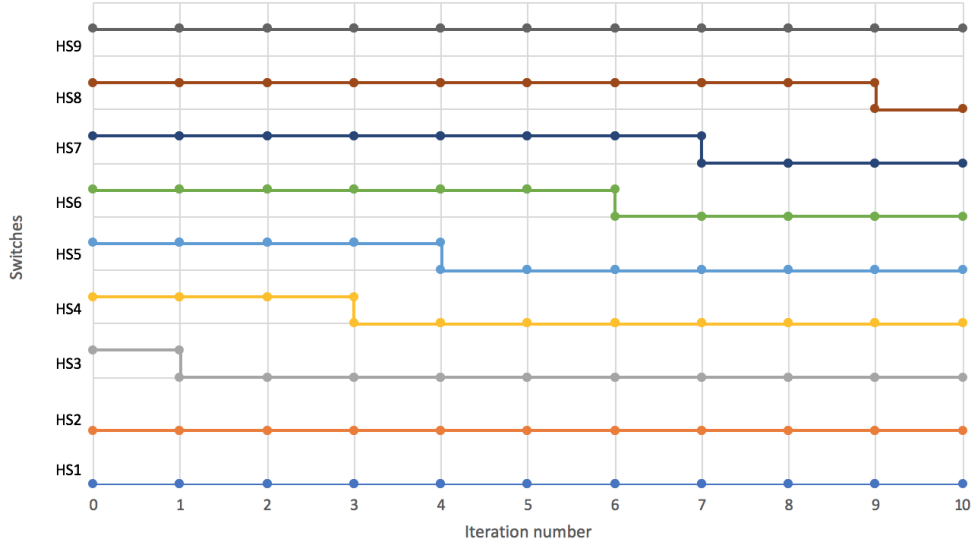


Figure 6.14: HS states for GO with decision based on delay values

Network value	Normalized value
$n = 10$	$n = 0.1$
$t = 1T_0^{-1}$	$t = 0.17$
$p = [0-40]\%$	$p = [0, 1]$
$d = 1 \text{ ms}$	$d = 0.01$

Table 6.3: Parameters for scenario 3

better than the baseline designs (fully centralized and fully distributed) at the boundaries. As expected, DO_{t_2} performs better than DO_{t_1} , since DO_{t_2} converges to centralization faster than DO_{t_1} .

6.2.3 Scenario 3: Effect of Rate of Unknown Packets

In this scenario, we study the effect of the quality of traffic on the network performance by increasing the % of corrupt packets on the data links. The rate of unknown packets varies between 0 and 40%. N is set to be 10, and the rest of the parameters are set to stable topology, and small delay to controller.

The parameters values are shown in table 6.3.

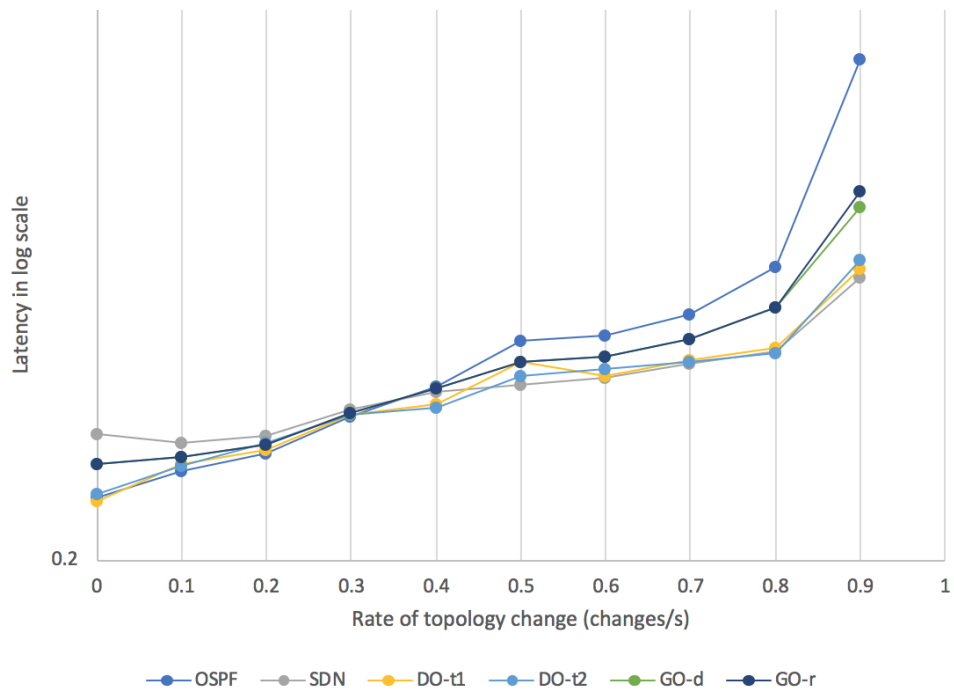


Figure 6.15: Latency with respect to rate of topology change for the different network designs in semi-log scale

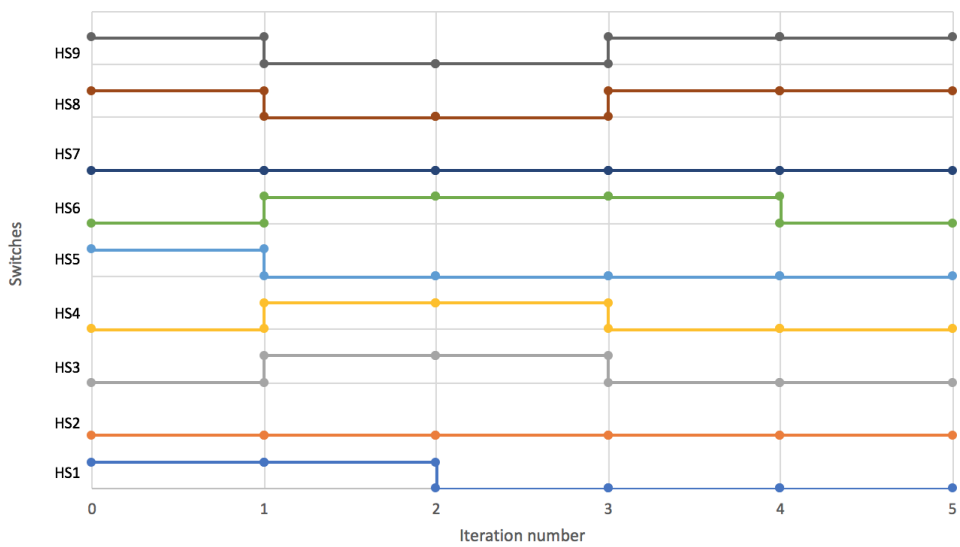


Figure 6.16: Switches states for DO with threshold t_1

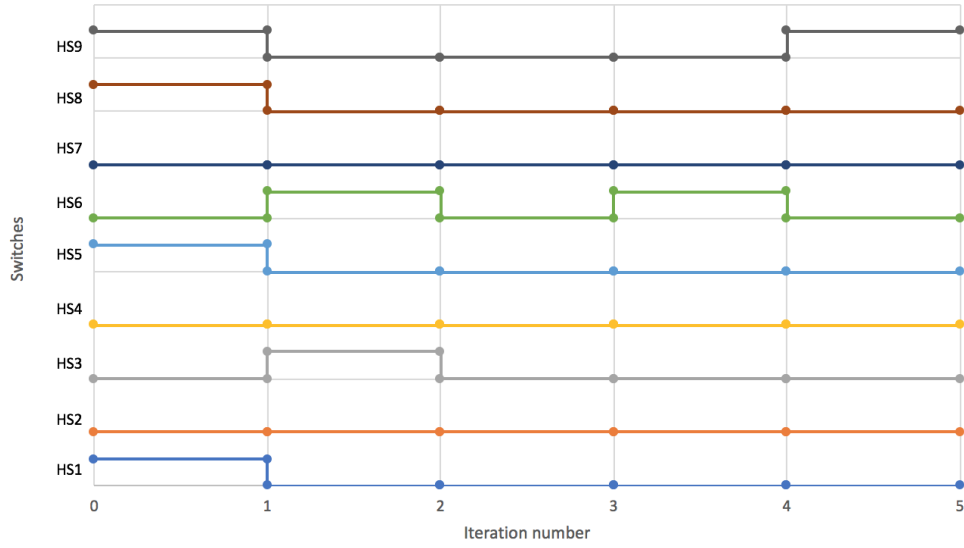


Figure 6.17: Switches states for DO with threshold t_2

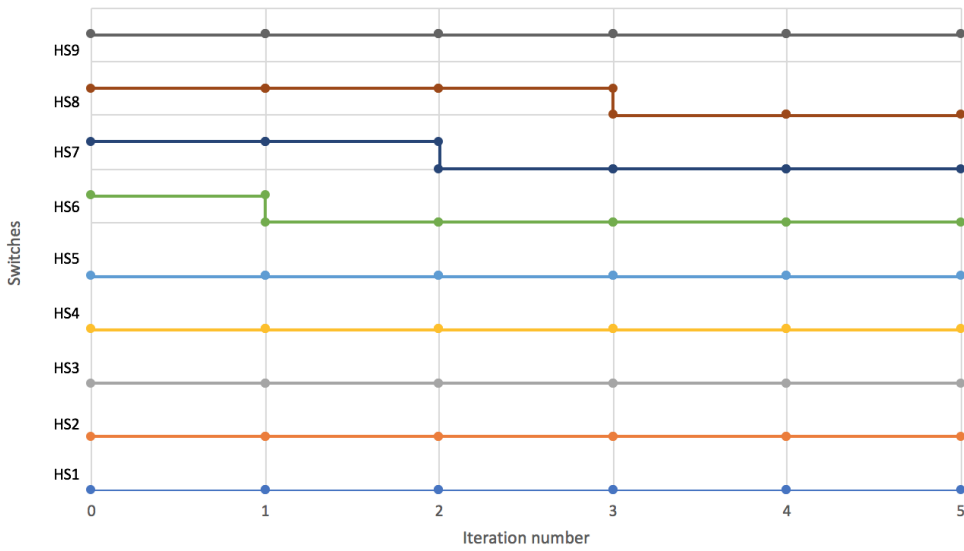


Figure 6.18: Switches states for GO with decision based on results of DO

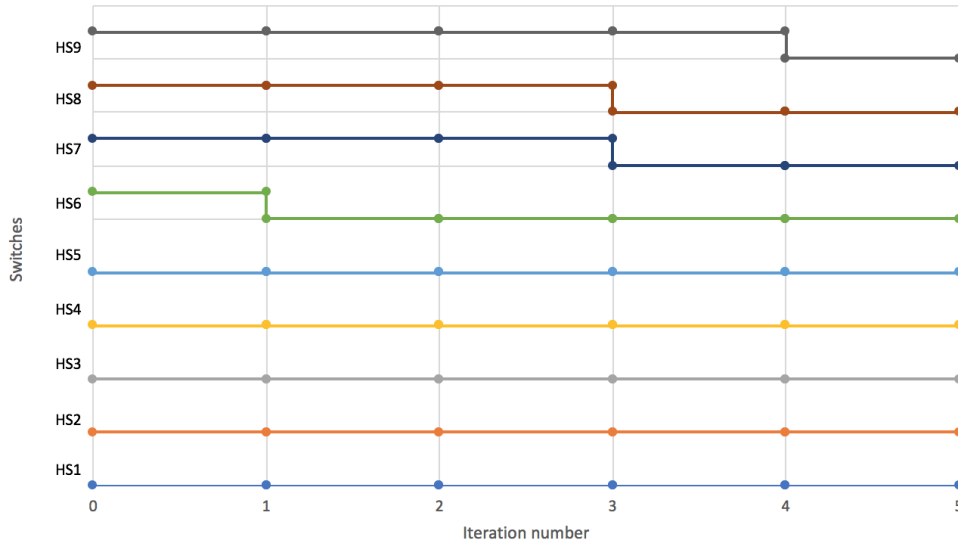


Figure 6.19: Switches states for GO with decision based on delay values

Figures 6.16 to 6.19 show the states of the HS's for the 4 optimization methods. We notice that in this scenario, the DO systems and the GO systems have comparable results when it comes to the states of the HS, and specifically the % of distribution in the network. When comparing the scenarios for the latency in Figure 6.20, we observe that the performance of OSPF is degraded with the increase in the % of corrupted packets; the performance of SDN is also affected but not to the same extent as OSPF. In fact, in the case of SDN, the decrease in performance is not due to its control plane, but rather to the data plane itself. OSPF is more affected mostly because the control plane and data plane share the same infrastructure, and if this latter is faulty, then not even data is corrupted, but also control information.

6.2.4 Scenario 4: Effect of Local Delay

In this scenario, we force the delay to controller to change suddenly for switches 8 and 9 at iterations 2 and 3, as shown in Figure 6.21.

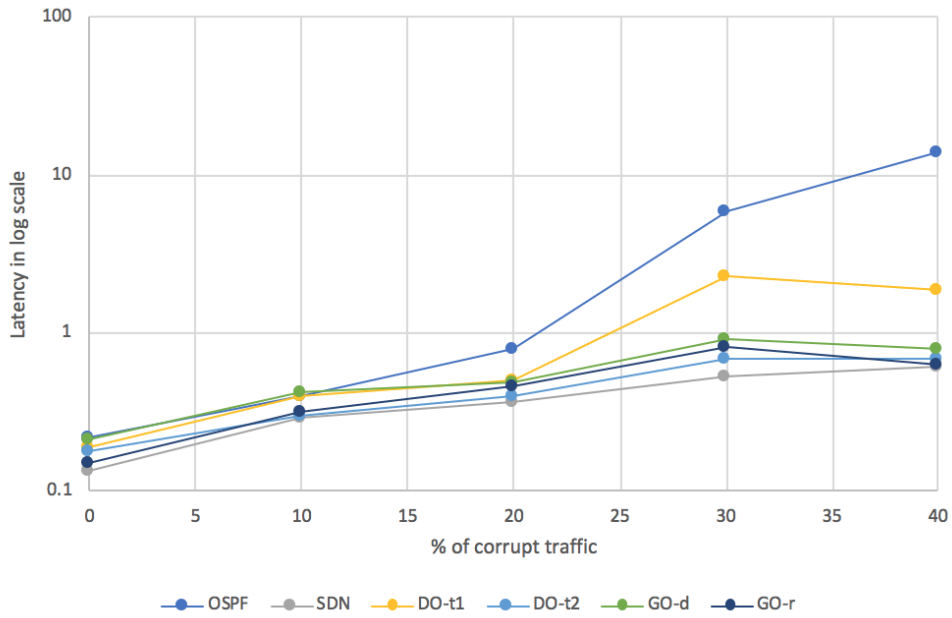


Figure 6.20: Latency with respect to rate of unknown traffic for the different network designs in semi-log scale

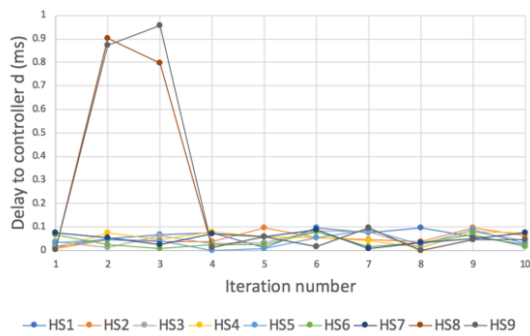


Figure 6.21: Delay between switches and controller

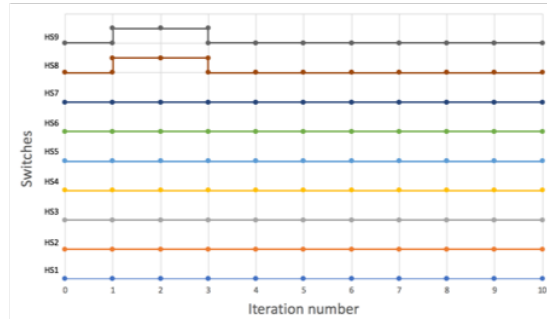


Figure 6.22: Switches states for DO with threshold t_1

We also tested the effect of the choice of the \mathbf{d} vector when used with GO; we considered it to be the maximum of the vector \mathbf{d} (GO-d(\mathbf{d}_{\max})), and the mean of \mathbf{d} (GO-d(\mathbf{d}_{mean})). We also consider DO with threshold t_1 , DO with threshold t_2 , and GO with decision based on the results of DO. Figures 6.22 to 6.26 show the states of the HS's for the 5 optimization methods. As expected, the DO algorithm with threshold t_1 detected the increase in delay and forced HS8 and HS9 to move to the distributed states for both iterations 2 and 3 (Figure 6.22). On the other hand, DO with threshold t_2 forced HS9 to go to the distributed states for both iteration, but only forced HS8 to go distributed for iteration 2 (Figure 6.23). This is because of the effect of t_2 on centralization.

When it come to the GO algorithm, we notice different behaviors for the GO-d(\mathbf{d}_{\max}) (Figure 6.24) and the GO-d(\mathbf{d}_{mean}) (Figure 6.25). In fact, GO-d(\mathbf{d}_{mean}) did not detect the sudden increase in delay for the 2 switches, whereas GO-d(\mathbf{d}_{\max}) did. This results show that GO algorithms in general are not sensitive to local issues such as a delay increase for 1 or 2 switches, and taking the average of the delays to controller attenuates the effect of the local sudden increase of the delay switches. This justifies and reinforces our 3rd design alternative which separates issues into local vs. global and assigns them to DO vs. GO respectively.

Figure 6.27 shows the latency of the network for each optimization technique

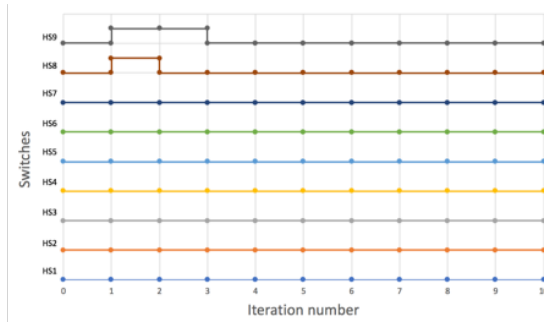


Figure 6.23: Switches states for DO with threshold t_2

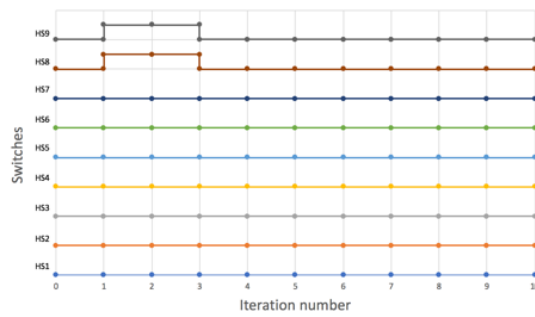


Figure 6.24: HS states for GO with decision based on maximum delay

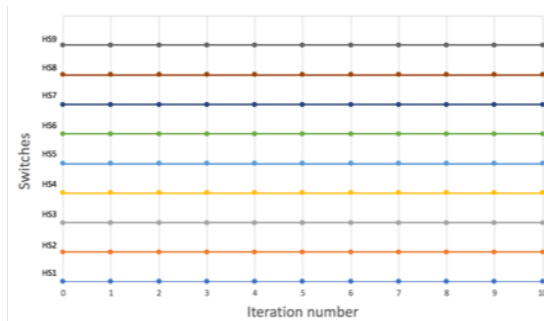


Figure 6.25: HS states for GO with decision based on mean delay

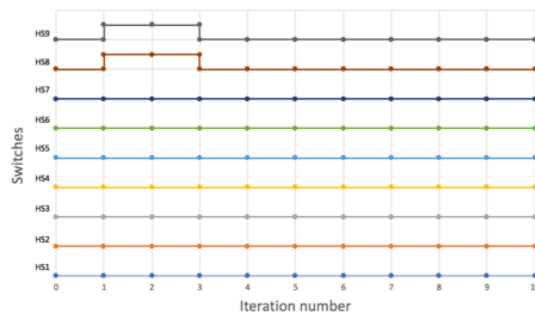


Figure 6.26: Switches states for GO with decision based on results of DO

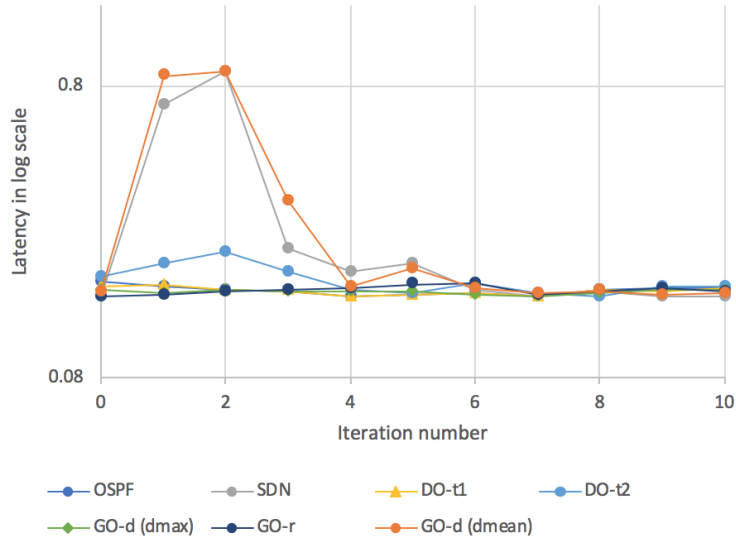


Figure 6.27: Latency for the different network designs in semi-log scale

for the values of the delay shown in Figure 6.21. We observe in this figure that all the optimizations that detected the delay increase for both iterations and for both switches have similar performance, and they perform better than full SDN and $GO-d(d_{mean})$. In fact, in the case of $GO-d(d_{mean})$, the network remained full SDN, and switches HS8 and HS9 were stuck waiting for a slow controller, which affected the performance of the algorithm at the second and third iteration as shown in Figure 6.27 (orange curve); this method incurred latency comparable to that of SDN. We notice that the latency of $DO-t_2$ slightly increased in the third iteration, due to the fact that this method forced HS8 to go centralised when its delay to controller was still high, which caused some extra latency in the network as compared to the other techniques that detected the delay for both iterations.

The above graphs (Figures 6.6 to 6.27) show the behavior of the optimizations systems and their effect on the network in comparison with SDN and OSPF. Tables 6.4 to 6.6 summarize the findings of the above graphs at two major points:

Delay to Controller	at 50%	at 100%
DO-t ₁	62.44%	89.13%
DO-t ₂	62.20%	88.93%
GO-d	59.15%	89.33%
GO-r	59.15%	89.13%

Table 6.4: Percentage improvement of proposed methods compared to baseline SDN at 50% and 100% of maximum delay to controller

they show in %, the improvement between the proposed methods and the baseline SDN/OSPF when the network condition under test in the given scenario is at 50% and 100% of its maximum allowed value.

Table 6.4 shows the percentage of improvement of each of the DO_{t1}, DO_{t2}, GO-d, and GO-r, with respect to SDN for scenario 1. In this case, we compare the latency of the 4 optimizations to SDN, since this latter is directly affected by the increase in delay to controller. Results showed that the proposed optimisations performed from 59% to 62% better than baseline SDN when the delay to controller is at 50% of its maximum allowed value, and they perform up to 89% better than SDN when the delay is at 100% of its max allowed value.

Additionally, table 6.5 shows the percentage of improvement of each of the DO_{t1}, DO_{t2}, GO-d, and GO-r, with respect to OSPF for scenario 2. In this case, OSPF is the technology that is directly affected by the increase in the rate of topology change, consequently we compare the latency of the 4 optimizations to OSPF. Results showed that the proposed optimisations also performed better than OSPF; at 50% of maximum rate of topology change, the proposed systems showed 5% increase in performance, whereas they showed an increase in performance of up to 29% for 100% of maximum rate of topology change.

Finally, table 6.6 shows the percentage of improvement of each of the DO_{t1}, DO_{t2}, GO-d, and GO-r, with respect to OSPF for scenario 3. Similarly to Table

Topology Change	at 50%	at 100%
DO-t ₁	3.47%	29.35%
DO-t ₂	5.56%	28.26%
GO-d	3.47%	21.74%
GO-r	3.47%	19.57%

Table 6.5: Percentage improvement of proposed methods compared to baseline OSPF at 50% and 100% of maximum rate of topology change

Traffic Corruption	at 50%	at 100%
DO-t ₁	37.50%	86.43%
DO-t ₂	50.00%	95.01%
GO-d	38.63%	94.29%
GO-r	41.63%	95.49%

Table 6.6: Percentage improvement of proposed methods compared to baseline OSPF at 50% and 100% of maximum rate of traffic corruption

6.6, we compare the latency of the 4 optimizations to OSPF because traffic corruption affects the control plane of OSPF and degrades its performance. Results showed that the proposed optimisations performed from 37% to 50% better than baseline OSPF when the rate of traffic corruption is at 50% of its maximum allowed value, and up to 95% better than OSPF when the rate of traffic corruption is at 100% of its max allowed value.

Chapter 7

Summary of Results and Analysis

7.1 Overall Results and Analysis

The introduction of centralized control to network introduce a new form of network that we were not used to before: a hybrid network that combines the centralized SDN paradigm and the existing distributed legacy network model. This type of network ought to exist not by design, but due to the incremental introduction of SDN islands in the old prevailing worldwide IP network. The first contribution in this thesis was to study hybrid networks and look into how SDN can support these hybrid networks, by looking into how the different network services, including routing, are performed in the hybrid forms of network. Additionally, given that these networks are harder to manage, we looked into network management with SDN. The design proposed and the results showed that even though the SDN controller can perform management functions, a separate SDN management layer in conjunction with the controller is essential to guarantee better network management. In fact, as the controller is in charge of supervising the network, running network applications, and instructing the switches, augment-

ing it with a management role risks overloading it, which would lead to network interruption. Not to mention that in hybrid networks, the management functionalities are much more complex because the legacy appliances need special integration within the SDN network.

The coexistence between the two types of network led us to the third contribution of this work, which is a study of systems that exhibit the same characteristics of two control planes, mainly the nervous system, and political administration systems. We studied these systems and concluded the control rules adopted in each to switch from centralization to decentralization and vice-versa. This study constituted a road-map for designing a hybrid system that shifts between its centralized and distributed states as per the extracted control rules.

In order to understand how IP and SDN network behave under different conditions, we consider one aspect of network performance - network convergence, which we modeled for the two systems. This work constituted the next contribution, as we set the mathematical model of convergence of both technologies in general networks, in WAN, and data-centers. We also analyzed their behavior both analytically and experimentally. Comparing analytical to experimental results allowed us to validate the general model. Essentially, experimental results show that convergence of OSPF is affected by the network link delays whereas SDN convergence is not, which matches the finding of the analytical testing. Also, when it comes to fault location, experimental results show that OSPF is more affected by the location of the fault than SDN, which is also in line with the analytical finding. Results from the experiments of WAN and data-centers weren't always in accordance with the results from the general model, which justifies the extension of the general convergence models, and validates our hypothesis that in addition to network parameters and circumstances, the performance does depend

on the network type and architecture.

The first hybrid system proposed was an offline recommender system based on fuzzy logic. This system takes as inputs parameters from the network, and proposes the appropriate type of technology to use given the network parameters along with the financial implications of the decision in terms of CAPEX and OPEX. This tool is an offline tool that is used by the network operator before deploying the network.

The next proposed designs aimed for online optimization tools that control the state of the network elements according to the prevailing network conditions. In this type of network, the network elements are hybrid switches that support centralized and distributed control planes and are allowed to change their control state between the two planes depending on the results of the optimization. The preliminary model developed a distributed optimization system, where each node runs its own optimization function and decide whether its control plane will rely on SDN or IP. Results showed the variations of the level of centralization vs. distributedness of the system with respect to changing network conditions correspond to the control model that we had specified based on guidelines in previous sections.

The final design consisted in designing a hybrid adaptive optimization system that controls the states of the hybrid network. The hybrid optimization system is composed of distributed and global optimization systems, that can operate as standalone systems but also together in a hybrid form as per the design alternatives specified earlier in this dissertation. Analytical results showed that the optimization design alternatives are in line with the control rules specified.

The last contribution of this work is the implementation of the hybrid network with all the related modules, and then running the final optimization design

alternatives to collect experimental results. The hybrid network was fully implemented in python, and functional tests showed that the network is running and responsive. Four optimization alternatives were tested on the hybrid network, and results showed that the optimal design outperform each other depending on some criteria. For example, at the boundaries, the DO system shows best performance, whereas when the conditions as a whole favor centralization, then the DO with threshold t_2 shows better performance. The GO systems also perform well, however, the testing is unfair because it does not take into consideration another aspect of the cost, which is the cost of running DO on all the network nodes each period, compared to running GO when the issues are global issues for example. Besides, when we compare SDN and OSPF, the performance is not only related to latency or packet loss, it is also related to the services made available with the chosen technology. For example, when NFV is needed, then SDN is favored as a technology, consequently having a non full OSPF network would be useful even if network conditions do not favor SDN completely. Also, the optimal designs push the network to adapt to the conditions and flip their states when SDN or OSPF become bottlenecks.

Chapter 8

Conclusions and Future

Directions

SDN provides better network management and network visibility with centralized control, whereas traditional distributed IP networks feature robustness and scalability. However, those characteristics sometimes change depending on network conditions. Controlled hybrid networks, as we define them, feature both types of control and exploit the advantages of each by adaptively changing the state of the network control plane to the most efficient control model given prevailing network conditions. The problem is generalized and modeled as an optimization problem that is designed according to rules extracted from different systems in political systems and natural intelligence. Additionally, the general system can be applied to other systems that exhibit the same kind of relationship between centralized and distributed control. We summarize the conclusions of this dissertation below.

8.1 Conclusion Related to Network Services and Management

After studying hybrid networks, we presented different network services and showed their operation in a traditional network, in SDN, and hybrid networks. Some services such as routing and switching have already been considered in all three types of networks, whereas some services like MPLS and multicasting are still being researched in SDN. We also analysed the services operation in the three schemes and postulates recommendations. Also, We investigated SDN management, and applied the FCAPS model to SDN and created a matrix of management functions with regards to FCAPS and the SDN controller/manager pair. Using this map, we classified management functions between controller and manager, and created an architecture for the SDN management Framework. A proof of concept was simulated to verify the performance gains of the proposed design which reached 27.5%.

8.2 Conclusion Related to Modeling Network Convergence

When it comes to convergence models, we modeled convergence in OSPF networks and SDN, and simulated the model in MATLAB. We then experimentally tested convergence delays of OSPF and SDN using Mininext. We obtained analytical results that match the theoretical expectations of both schemes. These results showed that SDN does provide convergence benefits under certain network conditions such as high network link delays, whereas the OSPF convergence benefits prevail under other conditions, for example when the SDN controller is

overloaded. The experimental study verified the validity of the theoretical model, with some irregularities due to the implementation details of the controller, the efficiency of its modules, and the emulation of the switches [121], that are not apparent to operators and users. This model was then extended to account for control plane delays, and compare the two for important network types, namely datacenters and WANs, in order to study the effect of the network type and parameters on the performance of the protocols. Results showed that OSPF and SDN behave differently in datacenters and WANs, and that SDN and OSPF have varying convergence speeds within the same network type. In other words, in datacenters like in WANs, the two protocols outperform each other for different values of network variables. Although one can prefer a technology over the other, it all comes down to the network design and characteristics which actually affect the performance of the protocols and deliver better convergence speeds.

8.3 Conclusion Related to the Proposed Designs and the Adaptive Optimization System

The first system we proposed was based on the fact that nowadays we face three forms of networks, with the techniques for operating each form. Based on the different network alternatives, a fuzzy system that guides network operators in their choice of network deployment schemes was presented. We recognize that every network deployment has several other constraints to consider in the design, including other types of inputs and additional inferences rules, which can be added to enhance the model and refine results. As described, the Fuzzy System provides insights to operators which facilitate the choice of networks to deploy. The financial implications were analyzed to allow management to assess the impact of the

decision on the CAPEX and OPEX. Subsequently, we designed an optimization system that adaptively decides on the control state of the network node (Hybrid Switches-HS), based on network conditions. The cost functions used are based on comparative performance results from the literature and our previous work. Results verified that the designed optimization provides the expected behavior of the model for the different conditions that we tested. The final design that we proposed introduces the notion of centralization and decentralization to the optimization formulation itself. It combines both Global Optimization (GO) and Distributed Optimization (DO), which act as stand-alone systems, fall-back systems, or complementary systems. The complete final design including the HSs, the controller module, the optimization systems, and the HS state monitoring and control systems were implemented on a variation of Mininet and the OSHI simulator. Results showed that the hybrid networks do provide good performance especially at the boundaries where the network parameters are critical to the correct operation of the network.

8.4 Future Directions

This work opens up to many future developments.

First of all, the implementation system can be improved by designing a merged flow table, that groups control information from both control planes. The merged flow table is a more efficient technique that will use rule translation to merge the rules from both control plane in one table and using the same syntax, as described in section 5.3.1. This enhancement will reduce delays further, by eliminating the need of switching from one flow table to the other and mapping physical to virtual ports on the HS. This will also reduce the incurred overhead due to packets

traversing the switch twice before being forwarded on the data path.

Also, the results can be improved by deploying the design on a real test-bed. In fact, the virtual machine and the implementation of the HS's in software incurs delays that wouldn't exist in real deployments. Consequently, latency measures in the results become more accurate.

When it comes to the hybrid design, the designed framework can also be used for service segregation: for example, instead of having nodes that are either centralized or distributed, the nodes can be centralized or distributed according to services, based on traffic tagging. In other words, a node can be configured to operate in legacy mode for one type of service such as normal routing, but go centralized for other types of services such as MPLS, based on the favorable control plane for each service (refer to section 3.2). In this case, the HS will route normal packets following any IP routing protocol, but rely on the controller's rules when it receives an MPLS encapsulated packet.

The decision making system can be improved. On one hand, the optimization can be extended to use a non-linear model by redefining the $h(s)$ functions that are currently used. On the other hand, introducing machine learning for decision making instead of optimization can yield a more dynamic and accurate response to network conditions and consequently better results. Employing machine learning also allows us to take advantage of the programmability feature of the SDN controller for the training of the algorithms in a centralized fashion.

Finally, the model can be applied to different areas. In fact, with the current

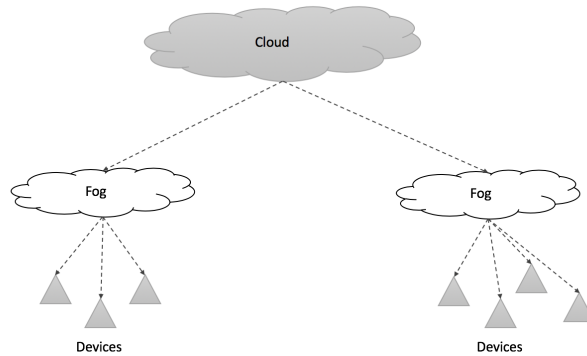


Figure 8.1: IOT network

increased need in scalability, performance, standardization, and virtualization, we are witnessing the introduction of centralized and decentralized instances of systems to cater for these requirements. In this context, the designed general model with its variables and parameters, can be applied to these systems.

An example of such systems would be IOT. In fact, in IOT, one can take advantage of fog computing to move the intelligence closer to the devices in a distributed fashion. A high-level architecture of an IOT network is shown in Figure

In analogy with our general model, network parameters can be mapped to:

- d is the delay between the Cloud and the Fog.
- t is the rate of the availability of the links between the two layers.
- p is the heterogeneity of the IOT devices.
- and n is the number of IOT devices in the network.

Another example outside of the telecom area would be the DCS-PLC control systems in plants [154], (refer to Figure 8.2). The DCS, or distributed control system, is composed of many distributed controllers that work autonomously.

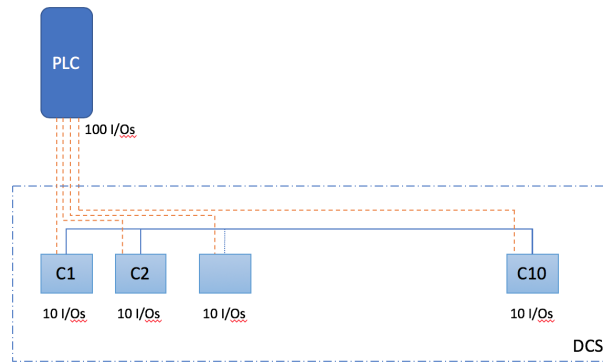


Figure 8.2: PLC-DCS system

The data is exchanged between the controllers but the system usually doesn't have overall central control. Alternatively, a PLC system, or the programmable logical controller system, is a centralized control system. In general, PLC's are more reliable in terms of availability and robustness, but DCS systems can scale for large systems, and implement more complex control loops. Consequently, both systems have their advantages; thus a hybrid design can be envisioned to deliver the best performance of both, and ensure availability in critical applications (for example in nuclear plants). In analogy with our general model, network parameters can be mapped to:

- d is the delay to compute control loops between the I/Os and the PLC.
- t is the rate of the availability of the links between the controllers.
- p is the I/O values integrity.
- and n is the number of I/Os in the system.

The application of the model to IOT and the PLC-DCS systems can be developed and tested to verify if the model does provide the intended results; also other areas can be identified to test the hybrid control model.

Bibliography

- [1] R. Musschebroek, “Sdn and nfv facilitate network convergence,” 2017.
- [2] G. Iannaccone, C.-N. Chuah, S. Bhattacharyya, and C. Diot, “Feasibility of ip restoration in a tier 1 backbone,” *Ieee Network*, vol. 18, no. 2, pp. 13–19, 2004.
- [3] A. S. Tanenbaum, *Routing Algorithms*, ch. 5.2. Computer Networks, Pearson Educación, 4th edition ed., 2003.
- [4] J. Doyle and J. D. Carroll, *Routing tcp/ip*, vol. 1. Cisco press, 2005.
- [5] C. Funakura, “Border gateway protocol best practices,” 2006.
- [6] C. E. Spurgeon and J. O. Zimmerman, *Basic Switch Operation*, ch. 1. Ethernet Switches, Charles E. Spurgeon and Joann Zimmerman, 2013.
- [7] O. N. Foundation, “Software-defined networking: The new norm for networks,” tech. rep., ONF White Paper, 2012.
- [8] D. Kreutz, F. M. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
- [9] O. N. Foundation, “Sdn architecture overview - version 1.0,” 2013.

- [10] D. Medhi and K. Ramasamy, *Network routing: algorithms, protocols, and architectures*. Morgan Kaufmann, 2017.
- [11] P. Lin, J. Bi, and H. Hu, “Asic: an architecture for scalable intra-domain control in openflow,” in *Proceedings of the 7th International Conference on Future Internet Technologies*, pp. 21–26, ACM, 2012.
- [12] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, “Balanceflow: controller load balancing for openflow networks,” in *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, vol. 2, pp. 780–785, IEEE, 2012.
- [13] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, *et al.*, “Onix: A distributed control platform for large-scale production networks.,” in *OSDI*, vol. 10, pp. 1–6, 2010.
- [14] A. Tootoonchian and Y. Ganjali, “Hyperflow: A distributed control plane for openflow,” in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, vol. 3, 2010.
- [15] S. Hassas Yeganeh and Y. Ganjali, “Kandoo: a framework for efficient and scalable offloading of control applications,” in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 19–24, ACM, 2012.
- [16] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, “Scalable flow-based networking with difane,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 351–362, 2011.

- [17] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “Devoflow: Scaling flow management for high-performance networks,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.
- [18] S. Vissicchio, L. Vanbever, and J. Rexford, “Sweet little lies: Fake topologies for flexible routing,” in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, p. 3, ACM, 2014.
- [19] D. Levin, M. Canini, S. Schmid, and A. Feldmann, “Incremental sdn deployment in enterprise networks,” in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 473–474, ACM, 2013.
- [20] S. Huang, J. Zhao, and X. Wang, “Hybridflow: A lightweight control plane for hybrid sdn in enterprise networks,” in *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pp. 1–2, IEEE, 2016.
- [21] B. Kar, E. H.-K. Wu, and Y.-D. Lin, “The budgeted maximum coverage problem in partially deployed software defined networks,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 394–406, 2016.
- [22] F. N. Farias, J. J. Salvatti, E. C. Cerqueira, and A. J. Abelém, “A proposal management of the legacy network environment using openflow control plane,” in *2012 IEEE Network Operations and Management Symposium*, pp. 1143–1150, IEEE, 2012.
- [23] C. Hall, D. Yu, Z. li Zhang, J. Stout, A. Odlyzko, A. W. Moore, J. Camp, K. Benton, and R. Anderson, *Collaborating with the enemy on network management*, pp. 154–162. Security Protocols XXII, Springer, 2014.

- [24] M. Caria, A. Jukan, and M. Hoffmann, “Sdn partitioning: A centralized control plane for distributed routing protocols,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 381–393, 2016.
- [25] V. Kotronis, X. Dimitropoulos, and B. Ager, “Outsourcing the routing control logic: Better internet routing based on sdn principles,” in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 55–60, ACM, 2012.
- [26] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, “Traffic engineering in sdn/ospf hybrid network,” in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pp. 563–568, IEEE, 2014.
- [27] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, “B4: Experience with a globally-deployed software defined wan,” *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.
- [28] M. Karakus and A. Durresi, “A survey: Control plane scalability issues and approaches in software-defined networking (sdn),” *Computer Networks*, vol. 112, pp. 279–293, 2017.
- [29] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, “Logically centralized?: state distribution trade-offs in software defined networks,” in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 1–6, ACM, 2012.
- [30] S. Vissicchio, L. Vanbever, L. Cittadini, G. G. Xie, and O. Bonaventure, “Safe update of hybrid sdn networks,” *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 3, pp. 1649–1662, 2017.

- [31] S. Vissicchio, L. Cittadini, O. Bonaventure, G. G. Xie, and L. Vanbever, “On the co-existence of distributed and centralized routing control-planes,” in *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 469–477, IEEE, 2015.
- [32] Mukhtar, “Centralization & decentralization.” <http://management4best.blogspot.com/2010/02/centralization-decentralization.html>, February 2010.
- [33] R. Garnaut, L. Song, and J. Golley, *China: The next twenty years of reform and development*. ANU Press, 2010.
- [34] R. A. Musgrave *et al.*, “Theory of public finance; a study in public economy,” 1959.
- [35] W. Oates, “Fiscal federalismharcourt brace jovanovich,” *New York*, 1972.
- [36] R. J. D. Figueiredo and B. R. Weingast, “Self-enforcing federalism,” *Journal of Law, Economics, and Organization*, vol. 21, no. 1, pp. 103–135, 2005.
- [37] B. R. Weingast, “The constitutional dilemma of economic liberty,” *The Journal of Economic Perspectives*, vol. 19, no. 3, pp. 89–108, 2005.
- [38] A. O. Bowman and R. C. Kearney, *State and local government*. Nelson Education, 2015.
- [39] W. F. Ganong and K. E. Barrett, *Review of medical physiology*. Appleton —& Lange Norwalk, CT, 1995.
- [40] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. Hudspeth, *Principles of neural science*, vol. 4. McGraw-hill New York, 2000.

- [41] M. Atzori, “Blockchain technology and decentralized governance: Is the state still necessary?,” *Available at SSRN 2709713*, 2015.
- [42] M. Crosby, P. Pattanayak, S. Verma, V. Kalyanaraman, *et al.*, “Blockchain technology: Beyond bitcoin,” *Applied Innovation*, vol. 2, no. 6-10, p. 71, 2016.
- [43] C. Fromknecht, D. Velicanu, and S. Yakoubov, “A decentralized public key infrastructure with identity retention,” *IACR Cryptology ePrint Archive*, vol. 2014, p. 803, 2014.
- [44] S. Mumtaz and J. Rodriguez, *Smart device to smart device communication*. Springer, 2014.
- [45] M. N. Tehrani, M. Uysal, and H. Yanikomeroglu, “Device-to-device communication in 5g cellular networks: challenges, solutions, and future directions,” *IEEE Communications Magazine*, vol. 52, no. 5, pp. 86–92, 2014.
- [46] S. Vissicchio, L. Vanbever, and O. Bonaventure, “Opportunities and research challenges of hybrid software defined networks,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 70–75, 2014.
- [47] C. Jin, C. Lumezanu, Q. Xu, Z.-L. Zhang, and G. Jiang, “Telekinesis: controlling legacy switch routing with openflow in hybrid networks,” in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, p. 20, ACM, 2015.
- [48] V. Kotronis, X. Dimitropoulos, and B. Ager, “Outsourcing the routing control logic: Better internet routing based on sdn principles,” in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 55–60, ACM, 2012.

- [49] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, and E. Salvadori, “Oshi-open source hybrid ip/sdn networking (and its emulation on mininet and on distributed sdn testbeds),” in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pp. 13–18, IEEE, 2014.
- [50] J. R. Vacca, *Computer and information security handbook*. Newnes, 2012.
- [51] R. Oppliger, “Internet security: firewalls and beyond,” *Communications of the ACM*, vol. 40, no. 5, pp. 92–102, 1997.
- [52] S. Hogg, “Is an sdn switch a new form of a firewall?.” <http://www.networkworld.com/article/2905257/sdn/is-an-sdn-switch-a-new-form-of-a-firewall.html>, 2015.
- [53] S. RAO, “Opendaylight is one of the best controllers for openstack — here’s how to implement it.” <http://thenewstack.io/opendaylight-is-one-of-the-best-controllers-for-openstack-heres-how-to-implement-it/>, 2015.
- [54] P. Floodlight, “Floodlight.” www.projectfloodlight.org.
- [55] R. project team, “Ryu sdn framework.” <https://osrg.github.io/ryu-book/en/html/index.html>.
- [56] M. Suh, S. H. Park, B. Lee, and S. Yang, “Building firewall over the software-defined network controller,” in *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, pp. 744–748, IEEE, 2014.
- [57] H. Hu, G.-J. Ahn, W. Han, and Z. Zhao, “Towards a reliable sdn firewall,” *Presented as part of the Open Networking Summit 2014 (ONS 2014)*, 2014.

- [58] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, “Extending sdn to handle dynamic middlebox actions via flowtags,” *Presented as part of the Open Networking Summit*, vol. 2014, 2014.
- [59] K. Thomas, ed., *Beginning Ubuntu Linux: From Novice to Professional*. Apress, 2006.
- [60] J. Anderson and J. Martin, “Towards a system for controlling client-server traffic in virtual worlds using sdn,” in *Proceedings of Annual Workshop on Network and Systems Support for Games*, pp. 1–2, IEEE Press, 2013.
- [61] H. Cho, S. Kang, and Y. Lee, “Centralized arp proxy server over sdn controller to cut down arp broadcast in large-scale data center networks,” in *Information Networking (ICOIN), 2015 International Conference on*, pp. 301–306, IEEE, 2015.
- [62] T. Porter, “The perils of deep packet inspection,” *Security Focus*, 2005.
- [63] W. Stallings, “Software-defined networks and openflow,” *The internet protocol Journal*, vol. 16, no. 1, pp. 2–14, 2013.
- [64] Y. Li and R. Fu, “An parallelized deep packet inspection design in software defined network,” in *Information Technology and Electronic Commerce (ICITEC), 2014 2nd International Conference on*, pp. 6–10, IEEE, 2014.
- [65] SANS-Institute, “Intrusion detection systems; definition, need and challenges,” 2001.

- [66] A. G. P. Lobato, U. da Rocha Figueiredo, and O. Duarte, “An architecture for intrusion prevention using software defined networks,” *Universidade Federal do Rio de Janeiro-GTA/COPPE-Rio de Janeiro, Brazil*, 2013.
- [67] O. Joldzic, Z. Djuric, and D. Vukovic, “Building a transparent intrusion detection and prevention system on sdn,” *Norsk informasjonssikkerhetskonferanse (NISK)*, vol. 7, no. 1, 2014.
- [68] Z. A. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, “Simple-fying middlebox policy enforcement using sdn,” in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 27–38, ACM, 2013.
- [69] Z. Qazi, C.-C. Tu, R. Miao, L. Chiang, V. Sekar, and M. Yu, “Practical and incremental convergence between sdn and middleboxes,” *Open Network Summit, Santa Clara, CA*, 2013.
- [70] L. D. Ghein, *MPLS Fundamentals*. Indianapolis, USA: Cisco Press, 2006.
- [71] pmoyer, “Openflow/sdn —& mpls, better together or mutually exclusive?,” 2012.
- [72] O. N. Foundation, “Openflow switch specification- version 1.4.0,” 2013.
- [73] J.-M. Goyeneche, “Multicast over tcp/ip howto.” www.tldp.org/HOWTO/Multicast-HOWTO.html#toc2, 2015.
- [74] W.-K. Jia and L.-C. Wang, “A unified unicast and multicast routing and forwarding algorithm for software-defined datacenter networks,” *Selected Areas in Communications, IEEE Journal on*, vol. 31, no. 12, pp. 2646–2657, 2013.

- [75] A. Iyer, P. Kumar, and V. Mann, “Avalanche: data center multicast using software defined networking,” in *Communication Systems and Networks (COMSNETS), 2014 Sixth International Conference on*, pp. 1–8, IEEE, 2014.
- [76] L. Bondan, L. F. Müller, and M. Kist, “Multiflow: Multicast clean-slate with anticipated route calculation on openflow programmable networks,” *Journal of Applied Computing Research*, vol. 2, no. 2, pp. 68–74, 2013.
- [77] M. Mahalingam, D. G. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, “Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks.,” *RFC*, vol. 7348, pp. 1–22, 2014.
- [78] K. Wanguhu, “Vxlan primer-part 1.” <http://www.borgcube.com/blogs/2011/11/vxlan-primer-part-1/>, November 2011.
- [79] B. Salisbury, “Configuring vxlan and gre tunnels on openvswitch.” <http://networkstatic.net/configuring-vxlan-and-gre-tunnels-on-openvswitch/>, July 2, 2012 2012.
- [80] D. Mahler, “Vxlan overlay networks with open vswitch.” <https://www.youtube.com/watch?v=tnSkHhsLqpM>, 2014.
- [81] A. S. Tanenbaum, *Quality of Service*, ch. 5.4. Computer Networks, Pearson Educación, 4th edition ed., 2003.
- [82] H. E. Egilmez and S. Civanlar, “An optimization framework for qos-enabled adaptive video streaming over openflow networks,” *Multimedia, IEEE Transactions on*, vol. 15, no. 3, pp. 710–715, 2013.

- [83] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelém, “Control of multiple packet schedulers for improving qos on openflow/sdn networking,” in *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pp. 81–86, IEEE, 2013.
- [84] K. Govindarajan, K. C. Meng, H. Ong, W. M. Tat, S. Sivanand, and L. S. Leong, “Realizing the quality of service (qos) in software-defined networking (sdn) based cloud infrastructure,” in *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*, pp. 505–510, IEEE, 2014.
- [85] ONF, “Open networking foundation.”
- [86] O. N. Foundation, “Sdn architecture overview - version 1.1,” 2016.
- [87] S. Schaller and D. Hood, “Software defined networking architecture standardization,” *Computer Standards & Interfaces*, vol. 54, pp. 197–202, 2017.
- [88] C. Rotsos, D. King, A. Farshad, J. Bird, L. Fawcett, N. Georgalas, M. Gunkel, K. Shiimoto, A. Wang, A. Mauthe, *et al.*, “Network service orchestration standardization: A technology survey,” *Computer Standards & Interfaces*, vol. 54, pp. 203–215, 2017.
- [89] J. A. Wickboldt, W. P. D. Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, “Software-defined networking: management requirements and challenges,” *Communications Magazine, IEEE*, vol. 53, no. 1, pp. 278–285, 2015.
- [90] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, “Frenetic: A network programming language,” in *ACM SIGPLAN Notices*, vol. 46, pp. 279–291, ACM, 2011.

- [91] A. Voellmy, H. Kim, and N. Feamster, “Procera: a language for high-level reactive network control,” in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 43–48, ACM, 2012.
- [92] S. Kuklinski, “Programmable management framework for evolved sdn,” in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–8, IEEE, 2014.
- [93] A. Devlic, W. John, and P. Skoldstrom, “A use-case based analysis of network management functions in the onf sdn model,” in *Software Defined Networking (EWSDN), 2012 European Workshop on*, pp. 85–90, IEEE, 2012.
- [94] Y. Wang and I. Matta, “Sdn management layer: Design requirements and future direction,” in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pp. 555–562, IEEE, 2014.
- [95] P. Smith, A. Schaeffer-Filho, D. Hutchison, and A. Mauthe, “Management patterns: Sdn-enabled network resilience management,” in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–9, IEEE, 2014.
- [96] A. Schaeffer-Filho, P. Smith, A. Mauthe, D. Hutchison, Y. Yu, and M. Fry, “A framework for the design and evaluation of network resilience management,” in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pp. 401–408, IEEE, 2012.
- [97] W. Han, H. Hu, and G.-J. Ahn, *LPM: Layered Policy Management for Software-Defined Networks*, pp. 356–363. Data and Applications Security and Privacy XXVIII, Springer, 2014.

- [98] P. Kazemian, “Header space library.” <https://bitbucket.org/peymank/hassel-public/wiki/Home>, July 2014.
- [99] A. Shalimov, D. Morkovnik, S. Nizovtsev, and R. Smeliansky, “Easy-way: simplifying and automating enterprise network management with sdn/openflow,” in *Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia*, p. 8, ACM, 2014.
- [100] S. Kuklinski and P. Chemouil, “Network management challenges in software-defined networks,” *IEICE Transactions on Communications*, vol. 97, no. 1, pp. 2–9, 2014.
- [101] S. Sasidharan and S. K. Chandra, “Defining future sdn based network management systems characterization and approach,” in *Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on*, pp. 1–5, IEEE, 2014.
- [102] P. Patil, A. Gokhale, and A. Hakiri, “Bootstrapping software defined network for flexible and dynamic control plane management,” in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5, IEEE, 2015.
- [103] F. S. Systems, “Fcaps,” 2005.
- [104] LinuxFoundation, “Opendaylight.” www.opendaylight.org.
- [105] P. Song, Y. Liu, T. Liu, and D. Qian, “Controller-proxy: Scaling network management for large-scale sdn networks,” *Computer Communications*, vol. 108, pp. 52–63, 2017.

- [106] H. Zhang and J. Yan, “Performance of sdn routing in comparison with legacy routing protocols,” in *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 491–494, IEEE, 2015.
- [107] D. Sankar and D. Lancaster, “Routing protocol convergence comparison using simulation and real equipment,” *Advances in Communications, Computing, Networks and Security*, vol. 10, pp. 186–194, 2013.
- [108] M. Abdulkadhim, “Routing protocols convergence activity and protocols related traffic simulation with it’s impact on the network,” *International Journal of Science, Engineering and Computer Technology*, vol. 5, no. 3, p. 40, 2015.
- [109] D. Pei, B. Zhang, D. Massey, and L. Zhang, “An analysis of convergence delay in path vector routing protocols,” *Computer Networks*, vol. 50, no. 3, pp. 398–421, 2006.
- [110] P. Lapukhov, “Ospf fast convergence, ine.” <http://blog.ine.com/2010/06/02/ospf-fast-convergenc/>.
- [111] D. Gopi, S. Cheng, and R. Huck, “Comparative analysis of sdn and conventional networks using routing protocols,” in *2017 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pp. 108–112, IEEE, 2017.
- [112] I. CNET Networks, “Understanding the protocols underlying dynamic routing.” <https://www.techrepublic.com/article/understanding-the-protocols-underlying-dynamic-routing/>.

- [113] Y. Tsegaye and T. Geberehana, “Ospf convergence times”, master’s thesis, department of computer science and engineering, chalmers university of technology, sweden.”
<https://pdfs.semanticscholar.org/200f/c654519d0be791b5a4523fc0226043bb4dfa.pdf>.
- [114] Ruhann, “Ospf convergence.” <https://routing-bits.com/2009/08/06/ospf-convergence/>.
- [115] V. Balakrishnan, “Graph theory (schaum’s outline),” 1997.
- [116] M. C. Golumbic, *Algorithmic graph theory and perfect graphs*, vol. 57. Elsevier, 2004.
- [117] D. A. Patterson and J. L. Hennessy, *Computer organization and design MIPS edition: the hardware/software interface*. Newnes, 2013.
- [118] S. Ghimire, R. Ghimire, and G. B. Thapa, “Mathematical models of mb/m/1 bulk arrival queueing system,” *Journal of the Institute of Engineering*, vol. 10, no. 1, pp. 184–191, 2014.
- [119] E. M. S. Springer Verlag GmbH, “Exponential distribution - encyclopedia of mathematics.”
- [120] U. o. S. C. Network System Lab, “Mininext.” <https://github.com/USC-NSL/miniNExT>.
- [121] M. Kuźniar, P. Perešini, and D. Kostić, “What you need to know about sdn flow tables,” in *International Conference on Passive and Active Network Measurement*, pp. 347–359, Springer, 2015.
- [122] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in

- campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
- [123] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turetli, “A survey of software-defined networking: Past, present, and future of programmable networks,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.
- [124] Cisco, “Enterprise networks: Practical differences in lan and wan sdn deployments.” <https://blogs.cisco.com/enterprise/enterprise-networks-practical-differences-in-lan-and-wan-sdn-deployments>.
- [125] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, “Sddc: A software defined datacenter experimental framework,” in *2015 3rd International Conference on Future Internet of Things and Cloud*, pp. 189–194, IEEE, 2015.
- [126] C. Buragohain and N. Medhi, “Flowtrapp: An sdn based architecture for ddos attack detection and mitigation in data centers,” in *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 519–524, IEEE, 2016.
- [127] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, “Data center network virtualization: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 909–928, 2012.
- [128] K. Golani, K. Goswami, K. Bhatt, and Y. Park, “Fault tolerant traffic engineering in software-defined wan,” in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 01205–01210, IEEE, 2018.

- [129] O. Michel and E. Keller, “Sdn in wide-area networks: A survey,” in *2017 Fourth International Conference on Software Defined Systems (SDS)*, pp. 37–42, IEEE, 2017.
- [130] R. Ahmed and R. Boutaba, “Design considerations for managing wide area software defined networks,” *IEEE Communications Magazine*, vol. 52, no. 7, pp. 116–123, 2014.
- [131] K. Phemius and M. B. Thales, “Openflow: Why latency does matter,” in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pp. 680–683, IEEE, 2013.
- [132] L. A. Zadeh, G. J. Klir, and B. Yuan, *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers*, vol. 6. World Scientific, 1996.
- [133] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, “On controller performance in software-defined networks,” in *Presented as part of the 2nd {USENIX} Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.
- [134] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, “Software-defined networking (sdn): a survey,” *Security and communication networks*, vol. 9, no. 18, pp. 5803–5833, 2016.
- [135] K. Casier, S. Verbrugge, R. Meersman, J. Van Ooteghem, D. Colle, M. Pickavet, and P. Demeester, “A fair cost allocation scheme for capex and opex for a network service provider,” in *Proceedings of CTTE2006, the 5th Conference on Telecommunication Techno-Economics*, 2006.
- [136] M. Schmidt, “Capital expenditure (capex) explained.” [//www.business-case-analysis.com/capital-expenditure.html](http://www.business-case-analysis.com/capital-expenditure.html), Retrieved August 10, 2016.

- [137] H. Levine, “The 2020 wan takes shape – sdn, virtualization, and hybrid wans.” <http://www.networkworld.com/article/2971214/wan-optimization/the-2020-wan-takes-whape-sdn-virtualization-and-hybrid-wans.html>.
- [138] B. Bouchon-Meunier, M. Dotoli, and B. Maione, “On the choice of membership functions in a mamdani-type fuzzy controller,” 1996.
- [139] Flowgrammable, “How to process a packet-in message.”
- [140] R. Pujar and I. Camelo, “Path protection and failover strategies in sdn networks,” *Inocybe Technologies*, *Open Networking Summit*, 2016.
- [141] M. Casado, N. Foster, and A. Guha, “Abstractions for software-defined networks,” *Communications of the ACM*, vol. 57, no. 10, pp. 86–95, 2014.
- [142] R. Iazard, “How to process a packet-in message.”
- [143] E. Borcoci, “Control plane scalability in software defined networking,” in *InfoSys 2014 Conference, Chamonix, France*, 2014.
- [144] B. Xiong, X. Peng, and J. Zhao, “A concise queuing model for controller performance in software-defined networks.,” *JCP*, vol. 11, no. 3, pp. 232–237, 2016.
- [145] E. [48] Modiano, “Introduction to queueing theory.”
- [146] H. Pham, “System reliability concepts,” in *System Software Reliability*, pp. 9–75, Springer, 2006.
- [147] Y. Belyaev and C. E.V., “Weibull distribution - encyclopedia of mathematics.”

- [148] A. Technologies, “Introduction to traffic planning.”
- [149] M. Osman, J. Núñez-Martínez, and J. Manges-Bafalluy, “Hybrid sdn: Evaluation of the impact of an unreliable control channel,” in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 242–246, IEEE, 2017.
- [150] “netem.” <https://wiki.linuxfoundation.org/networking/netem>.
- [151] “Floodlight rest api.” <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1>
- [152] “ryu.app.ofctlrest.” https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html.
- [153] “iperf - the ultimate speed test tool for tcp, udp and sctp.” <https://iperf.fr>.
- [154] J. La Fauci, “Plc or dcs: selection and trends,” *ISA transactions*, vol. 36, no. 1, pp. 21–28, 1997.

