ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Communications Engineering
(31^{st} cycle)

Control Plane in Software Defined Networks and Stateful Data Planes

By

Abubakar Siddique Muqaddas

******

Supervisor:
Prof. Paolo Giaccone

Doctoral Examination Committee:
Prof. Franco Callegati, Università degli studi di Bologna
Prof. Francesco Musumeci, Politecnico di Milano

Politecnico di Torino
2019

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Abubakar Siddique Muqaddas
2019

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to my family: my parents, my wife, my brother and my grandparents.

With their duas, Alhamdulillah, I achieved which I could never hope to do so...

Acknowledgements

First and foremost, I would like to thank Almighty Allah; all my achievements are due to Him.

I would like to extend my sincere gratitude to my supervisor Prof. Paolo Giaccone for his guidance throughout my Masters thesis and Ph.D research. I learned from his way of research, questioning every little nuance of the work at hand, leading to exceptional and quality work. His acumen and immense knowledge helped me gain the direction in our work. Indeed, the research skills I have learned from him will help throughout my life.

I would like to thank Prof. Andrea Bianco for his valuable comments and insights. I am grateful to him for providing an opportunity to carry out research at the High Performance Networks (HPN) group at the University of Bristol.

I would like to thank Miquel Garrich Alabarce, for his constant drive and extreme determination in realizing our idea. Indeed, the basic knowledge that I have received about the optical networks is due to him.

I express my gratitude to Prof. Reza Nejabati and Prof. Dimitra Simeonidou for hosting me at the HPN group.

I would really like to thank Ahsan Mahmood and Tianzhu Zhang, who motivated me and provided an exceptional company during my research. I would also like to thank German Sviridov and Janvi Palan for their collaboration.

I also thank Navdeep Uniyal, Dimitris Gkounis, Thierno Abdourahmane Diallo and Anderson Bravalheri, for being the best colleagues, and supporting me during my stay at HPN group.

Lastly, I express my gratitude to my parents, wife, brother and grandparents, who were supportive and kept me motivated throughout my Ph.D.

Abstract

Legacy telecommunication networks are complex to configure due to the strict coupling of data plane and control plane. In this regard, Software Defined Networking (SDN) is a disruptive technology, which decouples the data plane and control plane, and offers a programmable data plane controlled by a logically centralized SDN controller. This simplifies the network control and provides a complete network view at the SDN controller. One of the key applications of SDN is 5G networks where it is utilized in conjunction with Network Function Virtualization (NFV) to provide end-to-end network service orchestration. 5G networks are being introduced to cope with increasing bandwidth, massive connectivity and low latency demands. SDN is thus advantageous to 5G since it can be used to provision dynamic network connectivity in an automated way. SDN has come a far away since the inception of OpenFlow in 2008, however there are multiple avenues in the scope of SDN to be further investigated. This thesis specifically investigates the control plane of SDN, specifically the various interfaces present in the SDN control plane. Additionally, we present the concept of replicated states in a stateful data plane in the scope of SDN.

As a first contribution, we investigate the east-west interface which carries inter-controller traffic among SDN controllers, so that the SDN controllers have a consistent view of the network. We quantify the impact of the network related data stores (topology, flows and hosts) on the traffic exchanged among a cluster of ONOS controllers. We empirically derive the throughput equation for the inter-controller traffic as a function of the network topology (number of switches and links). We also present the impact of network flows on the inter-controller traffic, the amount of inter-controller traffic based on flow backup for some commercial OpenFlow switches is reported. Furthermore, we also discuss the impact of the host store; where we provide a lower and upper bound on the amount of traffic exchanged between the controllers due to addition of a host in the network.

As a second contribution, in the scope of south-bound interface of an SDN controller, we propose a novel approach using time-synchronized operations (TSO) using timestamps in south-bound extensions in software-defined elastic optical networks. We present an end-of line-scenario in which lightpath rerouting is inevitable to provision a new lightpath. We analytically show that TSO operations executed simultaneously reduce the lightpath disruption time by a minimum of 75% while rerouting, instead of an asynchronous (ASY) approach, which consists of multiple operations executed separately. Moreover, we present an experimental validation of our approach, where both techniques (ASY and TSO) exhibit close network performance indicators (e.g. OSNR, power budget, spectrum tilt) after the lightpath swapping.

As a third contribution, we utilize the north-bound interface of an SDN controller to provision network connectivity between VNFs in a network service in both inter-domain and intra-domain network service orchestration. For the inter-domain case, we present the 5G UK Exchange (5GUKEx), which is a lightweight network orchestration platform. While integrating various operator platforms, 5GUKEx instantiates network services across different operator domains (islands) and stitches the network services using a common inter-domain network infrastructure. We experimentally demonstrate that the 5GUKEx is lightweight, since it delegates the resource orchestration to the islands. For the intra-domain case, we extend the ETSI-compliant Open Source MANO (OSM) with a Transport-API based WAN Infrastructure Manager (WIM) connector. This allows connectivity between VNFs running on different PoPs over a heterogenous network.

As a fourth contribution, we present the concept of replicated states in a stateful data planes. We present the argument that instead of using a single state variable which causes congestion and increase of network traffic, replicas of the state must be used to reduce the amount of traffic. We present an ILP formulation of the optimal placement of state replicas and routing of flows through the closest state copy. To solve for larger network topologies, this is accompanied by a heuristic.

Contents

Acronyms

ABW	Available bandwidth
AQMP	Advanced Queuing Messaging Protocol
ASY	Asynchronous
B&S	Broadcast-and-select
DISCO	Distributed Multi-domain SDN Controllers
EFSM	Extended Finite State Machines
eMBB	Enhanced Mobile Broadband
EON	Elastic Optical Networks
EPN	Electronic packet networks
FAs	Federation Agents
FMs	Federation Managers
FS	Frequency slots
GMPLS	Generalized Multi-Protocol Label Switching
IA-NC	Intelligent Agent NETCONF
IDCM	Inter-Domain Connectivity Manager
ILP	Integer Linear Programming
IoT	Internet of Things
LLDP	Link Layer Discovery Protocol
LPH	Fast lightpath hopping
LXC	Linux containers
MANO	Management and Orchestration
MbB	Make-before-Break
MdOs	Multi-domain Orchestrators
MEMS	Micro-Electro-Mechanical systems
mMTC	Massive Machine Type Communications

N-NC	Native-NETCONF
NBI	North Bound Interface
NFV	Network Function Virtualization
NFVO	NFV Orchestrator
NIB	Network Information Base
NS	Network service
NSDs	Network Service Descriptors
NSM	Network Service Manager
NSR	Network Service Record
OBS	One-big switch
ODL	OpenDaylight
OF	OpenFlow
ONOS	Open Network Operating System
OPP	Open Packet Processor
OS	Operating System
OSM	Open Source MANO
OSNR	Optical signal-to-noise ratio
OTCC	Open Transport Configuration & Control
OXC	Optical cross connect
OXM	OpenFlow Extensible Match
PCE	Path Computation Element
PoPs	Point-of-Presence
PTP	Precision Time Protocol
QoS	Quality of Service
RMT	Reconfigurable Match Tables
RO	Resource Orchestrator
ROADMs	Reconfigurable optical add/drop multiplexers
RSA	Routing and spectrum assignment
SDN	Software Defined Networking
SMF	Single mode fiber
T-API	Transport API
TSO	Time-synchronized operations
uRLLC	Ultra Reliable Low Latency Communications
VIM	Virtual Infrastructure Manager
VM	Virtual Machine

VNF	Virtual Network Function
VNFDs	VNF descriptors
VR	Viewstamped Replication
WAN	Wide-Area Networking
WDM	Wavelength division multiplexed
WIM	WAN Infrastructure Manager
WSS	Wavelength selective switch

Chapter 1

Introduction

Legacy telecommunication networks are complex and hard to reconfigure. In this case, Software Defined Networking (SDN) is a major innovation which has various benefits over the legacy networks. This thesis focuses on novel applications and analysis of the various interfaces as part of the SDN control plane, along with a novel proposal of utilizing replicated states in stateful SDN data planes.

This chapter introduces the concept of SDN, its architecture including the various control plane interfaces. Furthermore, the structure of this thesis is presented while briefly introducing the work conducted for each interface of the SDN controller and the stateful data planes.

1.1 Software Defined Networking

As mentioned, legacy telecommunication networks are complex to manage. For implementing a network-wide policy, each device needs to be configured separately in its own vendor-specific way, which is quite tedious and leads to high OPEX [1]. Furthermore, they are not able to sustain dynamic network behaviour and increasing load challenges, which is a distinct feature of the upcoming 5G networks. Another challenge that the legacy networks inhibit is the tight coupling of the control plane and data plane. Here, the control plane refers to the mechanism which decides the network policy, i.e., where to forward the traffic. The data plane consists of the network devices which actually forward the traffic based on the control plane decisions. This tight coupling of the control plane and data plane reduces flexibility

and increases the difficulty of implementing new network protocols and technologies. This challenge has been answered in the form of SDN.

In the last decade, SDN has emerged as a promising technology in telecommunication networks [1][2]. It has challenged the legacy control of networks by introducing the logically centralized SDN controller, which being essentially the "brain" of the network, has an entire view of the network under control. The control plane is decoupled from the network devices and the control logic is shifted to an SDN controller, while still allowing the network devices to forward the traffic. This creates the concept of a programmable data plane, where the data plane consists of simple forwarding devices and the forwarding policy is programmed onto the data plane by the SDN controller. In the following, the architecture of SDN is described.

1.1.1 SDN Architecture Overview

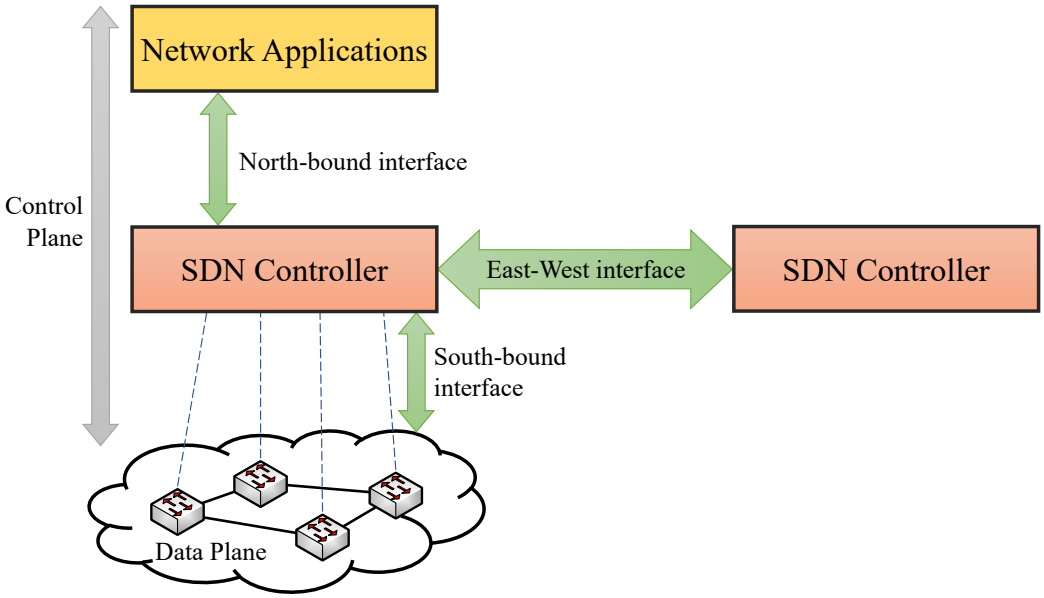


Fig. 1.1: SDN architecture

The architecture of SDN is depicted in Fig. 1.1. Here, we highlight some important components of the SDN architecture relevant for the thesis.

Control Plane

- **SDN controller:** SDN controller is the "brain" of the network. All the control logic related to enforcing network policies is embedded in it. It has the entire view of the network and provides a logically centralized control over the data plane, courtesy of the south-bound interface which is connected to data plane. It provides necessary abstraction of the data plane for the policy makers and network developers to enforce their relevant network policy via network applications utilizing the north-bound interface of the SDN controller. Examples of SDN controllers are ONOS [3] and OpenDaylight (ODL) [4].

- **South-bound interface:** This interface connects the control plane (SDN controller) to the data plane. It allows the SDN controller to enforce network policies on the data plane using well established south-bound protocols. SDN controller receives network related events from the data plane via this interface. Some well known south-bound protocols are OpenFlow [2] and NETCONF [5].

- **North-bound interface:** This interface is offered by the SDN controller to developers for their network applications which, for example, may range from basic packet forwarding to advanced multi-layer network slice provisioning. The SDN controller abstracts the low-level information about the data plane and offers its holistic network view to the network applications using the north-bound interface. The network applications can in turn use the north-bound interface to implement their selected policy on the data plane.

- **East-West interface:** A logically centralized SDN controller refers to the fact that there can be multiple distributed SDN controllers; however they all have the same consistent view of the network. This is enabled by the the east-west interface on which the SDN controllers exchange information about the data plane and network applications among each other. Furthermore, multiple SDN controllers are required in the network for redundancy; consequently one SDN controller is a master for a data plane device and other SDN controller(s) may act as backup slave controller(s).

Programmable data plane

The programmable data plane consists of many network elements which forward traffic, be it wired or a wireless network. The forwarding policy of the data plane is dictated by the SDN controller, hence the word "programmable". A data plane network element typically consists of the traffic forwarding mechanism along with a software agent which talks to the SDN controller via the south-bound interface. A modification of the conventional SDN data plane is the stateful data plane.

- **Stateful data plane:** It consists of memory inside a data plane switch which allows to maintain persistent states. This allows the data plane to perform complex in-network per-packet processing without relying on the SDN controller to make decisions. This improves the reactivity time by reducing latency, which was previously caused by the interaction with the SDN controller. The data plane switches can now take local decisions based on their internal states. Examples of stateful data plane are P4 [6] and Open Packet Processor (OPP) [7].

1.1.2 Benefits and Applications

The biggest advantage that SDN offers is the logically centralized view over the entire network; almost all of the applications which can benefit from SDN are due to this fact. The entire view of the network can offer simple implementation of network policies which are too complex to implement using legacy networks. A network developer can dictate his/her policy using the SDN controller and this policy is implemented over the entire network. This characteristic of SDN enables simultaneous control over various heterogeneous networks, spanning various technological domains (wired electronic packet/optical or wireless). Additionally, with combination of a programmable data plane, an SDN controller can provision connectivity while optimizing the network resources.

SDN is one of the key technologies in the upcoming 5G networks [8]. 5G networks are being developed as the next revolution in the telecommunications network, to cope with increasing demand for bandwidth and connectivity for an enormous number of devices. This entails the three key abilities of 5G: enhanced mobile broadband (eMBB) with data rates up to 10 Gbps to end users, ultra reliable

low latency communications (uRLLC) with round trip times as low as 1 ms, and massive Machine Type Communications (mMTC) e.g., Internet of Things (IoT) [9]. Since 5G supports massive number of connected devices, this will require dynamic and flexible network service provisioning; consequently the legacy network control systems are not able to sustain this. SDN thus offers a promising alternative, since it can provide automated network control and the ability to dynamically change the behavior of network devices as a response to the increasing/decreasing network service requests.

Another concept arising in 5G networks is Network Function Virtualization (NFV) [10]. In this concept, Virtual Network Functions (VNFs), which may run in Virtual Machines (VMs), containers or micro services are used to perform specific network related functions e.g., firewall, intrusion detection systems, load balancer etc. VNFs are easier and much flexible to deploy in commodity hardware, instead of setting up a specific hardware-based network middlebox. In this regard, the combination of SDN and NFV can provide virtualized network services on demand; where VNFs, as part of a network service, are interconnected leveraging SDN capabilities.

1.2 Structure of thesis

The focus of this thesis is the novel applications and analysis of the control plane of SDN, where its chapters focus on the innovative work involving the various interfaces present in an SDN controller as well as the stateful data plane. A sketch of the thesis organization is shown in Fig. 1.2. It refers to the control plane interface of an SDN controller as well as the data plane and corresponding chapter in the thesis. Furthermore, we briefly introduce the work conducted in the thesis.

1.2.1 Inter-controller traffic in distributed ONOS controllers

Chapter 2 is related to the east-west interface.

In distributed SDN architectures, the network is controlled by a cluster of multiple controllers. This distributed approach permits to meet the scalability and reliability requirements of large operational networks. Despite that, a logical centralized view

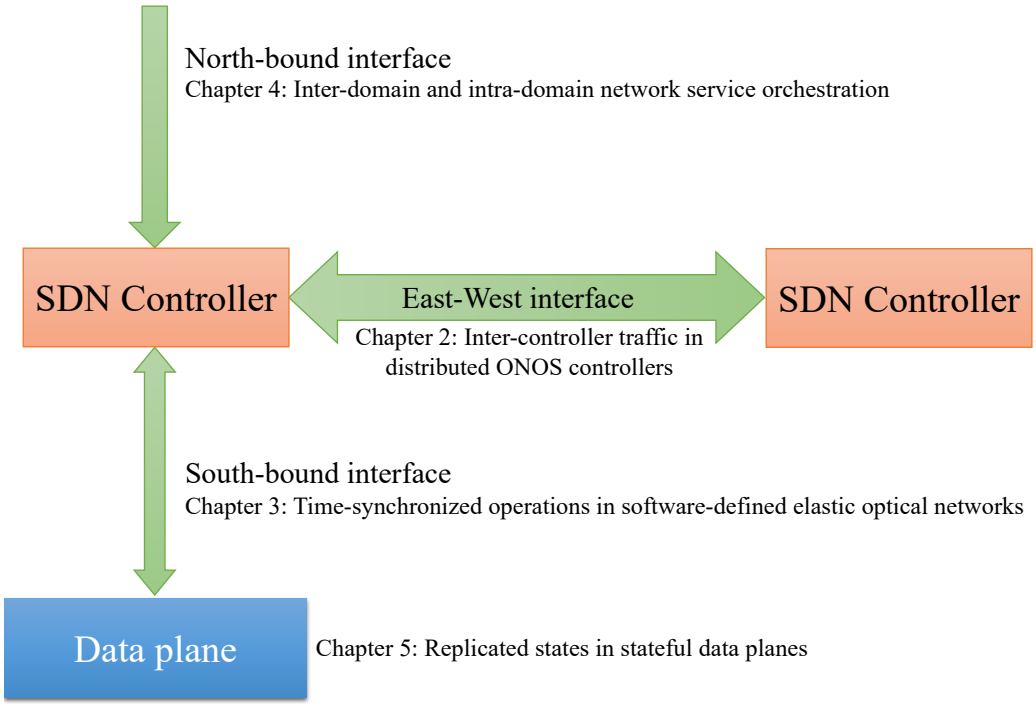


Fig. 1.2: Structure of the Thesis

of the network state should be guaranteed, enabling the simple development of network applications. Achieving a consistent network state requires a consensus protocol, which generates control traffic among the controllers whose timely delivery is crucial for network performance. This control traffic between the controllers is exchanged using the East-West interface as shown in Fig. 1.2.

We focus on the state-of-art ONOS controller, designed to scale to large networks, based on a cluster of self-coordinating controllers. In particular, we study the inter-controller control traffic due to the adopted consistency protocols. Based on real traffic measurements and the analysis of the adopted consistency protocols, we develop some empirical models to quantify the traffic exchanged among the controllers, depending on the considered shared data structures, the current network state (e.g., topology) and the occurring network events (e.g., flow or host addition). Our models provide a formal tool to be integrated into the design and dimension the control network interconnecting the controllers. Our results are relevant for the proper design of large SDN networks, in which the control plane is implemented in-band and cannot exploit dedicated network resources.

1.2.2 Time-Synchronized Operations for Software-defined Elastic Optical Networks

Chapter 3 is related to the south-bound interface.

Elastic Optical Networks (EON) have been proposed as a solution to efficiently exploit the spectrum resources in the physical layer of optical networks. Moreover, by centralizing legacy Generalized Multi-Protocol Label Switching (GMPLS) control-plane functionalities and providing a global network view, SDN enables advanced network programmability valuable to control and configure the technological breakthroughs of EON.

Here, we present our proposal of time-synchronized operations (TSO) to minimize disruption time during lightpath reassignment in EON. TSO have been recently standardized in SDN south-bound protocols NETCONF and OpenFlow; and here we discuss its implementation in optical networks. Subsequently, we update our analytical model considering an experimental characterization of the WSS operation time. Then, we present an experimental validation of TSO for lightpath reassignment in a five-node metropolitan optical network test-bed. Results validate the convenience of our TSO-based approach against a traditional asynchronous technique given its reduction of disruption time while both techniques maintain a similar network performance in terms of optical signal-to-noise ratio and optical power budget.

1.2.3 Inter and intra-domain network service orchestration

Chapter 4 is related to the north-bound interface.

We target network orchestration, which is the provisioning of a network service consisting of VNFs connected in a chain where the networking between the VNFs hosted at different Point-of-Presence (PoPs) is done using an SDN controller. Specifically, a service orchestrator, which is responsible for instantiating the network service, requests the SDN controller's north-bound interface to provision connectivity between the different PoPs. Our contribution has covered both inter-domain and intra-domain network orchestration as detailed below.

Inter-domain network orchestration

5G networks envisage to support a range of vertical industries, circumventing any potential barriers from converging various network technologies and administrative domains. Current solutions, such as the Management and Orchestration (MANO) standards and systems, which simultaneously orchestrate both network and compute resources using SDN and NFV respectively, focus only on provisioning services within single administrative domains. There is also lack of standards for end-to-end multi-domain coordination and sustainable multi-domain solutions that can use existing MANO systems. This is important to enable operators that control programmable infrastructures to collaborate with each other. In this thesis, we present the 5GUK Exchange (5GUKEx), a novel hierarchical architecture to enable end-to-end orchestration and coordination with minimum overhead in complexity and performance while also allowing operators to maintain full control of their infrastructure and integrate using their existing MANO systems. The key idea to allow operators to use their existing MANO systems for the single domain orchestration and building a multi-domain API is based on standardized models of service catalogs that MANO systems implement to coordinate the end-to-end service orchestration and interconnection. We build a prototype of the 5GUKEx and evaluate its performance through emulations showing that the 5GUKEx introduces minimum overhead.

Intra-domain network orchestration

As mentioned, 5G networks are ushering a new era of telecommunications, realizing network services which have high bandwidth and low latency demands. Furthermore, with the advent of NFV, VNFs are replacing hardware based network functions. VNFs are more flexible and can be deployed close to the customer at the edge to provide low latency services. However, VNFs hosted at different data centers within a single administrative domain need to be chained to create an end-to-end network service. Low latency inter-data center communication can be provisioned using optical networks which are controlled using a WAN Infrastructure Manager, which is essentially an SDN controller. Here, MANO systems play a vital role which permit simultaneous orchestration of compute and network resources. In this thesis, we experimentally demonstrate the integration of Transport API based WAN

Infrastructure Manager with Open Source MANO (OSM), for NFV orchestration over optical networks within a single administrative domain.

1.2.4 Replicated states in stateful data plane

Chapter 5 is related to the stateful data plane.

As briefly mentioned in Sec. 1.1.1, in stateful SDN data planes, switches hold some local state and can take local decisions. This is advantageous since only relying on the SDN controller in a stateless data plane can increase the delay in making decisions. This is highly important in mission critical applications where e.g., the data plane can reroute the traffic using an alternative route and not wait for the SDN controller to compute an alternative route and install forwarding rules on the devices; where the latter is a slower process.

We discuss the recently proposed SNAP framework [11], which jointly addresses the placement of the local states and the flow routing problem to minimize the total network data traffic while guaranteeing that all flows traverse the switches storing the flow related states. However SNAP assumes one single copy of each state: this limits the scalability of SNAP in case of states with a global scope. To overcome this limitation, we extend SNAP to support state replication, i.e. distribution of multiple copies of the same state across the available programmable switches. *TODO:* The following has been changed after Reviewer 1 comments. We provide the problem formalization to find the optimal placement of state replicas and the minimization of the total traffic in the network; this includes the data traffic and the state synchronization among the state replicas. We propose an approximation algorithm to solve the problem in large networks for a single state replication. We show the beneficial effect of state replication in a set of benchmark graphs.

Part I

Control Plane in SDN

Chapter 2

Inter-controller traffic in distributed ONOS controllers

Part of the work presented in this chapter has been published in:

- A. S. Muqaddas, P. Giaccone, A. Bianco and G. Maier, "Inter-controller traffic in ONOS clusters for SDN networks". In: *IEEE International Conference on Communications (ICC)*. May 2016, pp. 1–6.
- A. S. Muqaddas, P. Giaccone, A. Bianco and G. Maier, "Inter-Controller Traffic to Support Consistency in ONOS Clusters". In: *IEEE Transactions on Network and Service Management* 14.4 (Dec. 2017), pp. 1018–1031.

2.1 Motivation

A naïve centralized approach for SDN is based on a single controller managing all network switches. Even if this simplifies the network management and the development of network applications, it poses severe limitations to network scalability and reliability. Indeed, a single centralized controller is a single point of failure. Moreover, a single controller may not be able to handle a large number of switches, because communication load and processing overhead for the controller increases with the number of switches. Finally, in very large networks (as in WANs), switches can be physically very far from the controller, and due to the propagation delays, flow modifications in switches can experience large latency.

Distributed SDN controllers face all the above impairments. Multiple instances of the controller manage the whole network, which is divided into different domains, each of them under the control of one controller instance. Distribution of the controller functions over multiple physical servers improves the robustness of the control plane, by providing backup control resources for each network node. Furthermore, large networks can be handled, because the switch control is distributed among the controllers and the processing load can be balanced. Finally, being the control servers also geographically distributed across the network area, they can reduce the switch-to-controller delay, thus improving the controller reactivity as perceived by the network nodes.

However, a logical centralized view of the network state must be guaranteed also with distributed controllers, to ease the development of advanced network applications. This transparent behavior for the network operator/programmer comes at the cost of keeping all the shared data structures synchronized among the controllers by means of some consensus protocol. For example, the same network topology must be known at each controller to take correct routing decisions. However, since each controller is responsible for a subset of switches, it is of paramount importance to distribute any data plane related event in a timely fashion to keep the same state among the controllers and avoid possible misbehaviors (e.g., routing loops, firewall leaks), as highlighted in [12].

In large SDN networks (as SDWANs), the control plane distributed among the controllers is implemented in-band, without the possibility of relying on a dedicated out-of-band high-performance network as the data center scenarios [13]. This poses technical challenges in designing the control network, which does not only interconnect the switches to controllers, but also supports the communication between controllers. Due to the complexity of the adopted consensus protocols, the reactivity of the controllers as perceived by the switches depends also on the bandwidth and delays experienced in the inter-controller communication. This fact advocates a proper design and plan of the network supporting the control traffic, in particular guaranteeing adequate bandwidth for the inter-controller traffic.

We focus on the control traffic exchanged among the controllers, which is often neglected in the literature. We consider the state-of-art ONOS controller [14], which is supported by a large community of network operators and vendors. Differently from the initial versions of well-known ODL project [4], ONOS has been designed

specifically to cope with reliability and scalability issues arising in large ISP/WAN networks. It natively supports a distributed version of the controller, running on a cluster of servers.

2.1.1 Our contributions

We run an experimental testbed which includes a cluster of ONOS controllers and evaluate the amount of traffic exchanged among the controllers. Since the traffic depends on the specific updates committed on the shared data structure, we address our problem by analyzing the impact of each update for all the shared data structures (i.e. topology, flow and host stores) that manage the network state. Thanks to tailored experiments, we evaluate the exact amount of traffic in function of the specific event or change of state in the network and thus we develop some empirical models of the ONOS inter-controller traffic. Our results are general in terms of network topology and partition of the network into different controller domains. The adopted methodology is also general and provides experimental guidelines to extend our results to an arbitrary number of SDN controllers.

2.1.2 Organization of the chapter

The remainder of this chapter is organized as follows. Sec. 2.2 introduces the general architecture of distributed SDN controllers and describes the two main consistency models adopted to synchronize the data structures. We concentrate on the specific distributed architecture of ONOS and describe the two main protocols to achieve the consensus on the data structures. In Sec. 2.3, we present the methodology we adopt to quantify the impact of network related events on the inter-controller traffic for a general distributed cluster of SDN controllers. The subsequent three sections are devoted to investigate the impact of updates occurring in different shared data structures. Indeed, in Sec. 2.4, we concentrate on the store describing the network topology. The experimental data allows us to devise a set of empirical models to estimate the throughput of the inter-controller traffic, which we expect to hold for a broad set of topologies (Properties 1 and 2). In Sec. 2.5, we concentrate on the store describing the flow tables and investigate the impact of flow modifications in the switches. Finally, in Sec. 2.6 we concentrate on the store recording the hosts

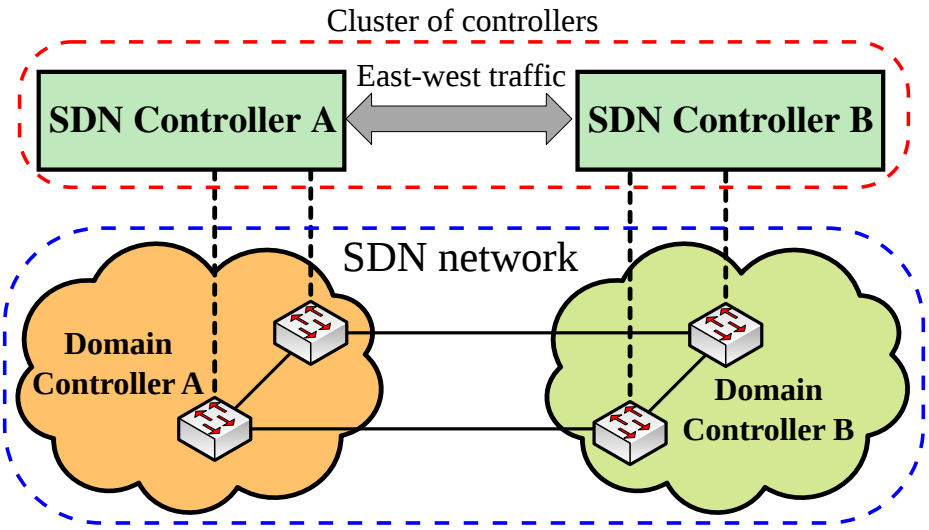


Fig. 2.1: Distributed SDN architecture with a single cluster of two controllers.

attached to the network switches. In Sec. 2.7 we discuss some related work, and finally in Sec. 2.8, we summarize the work presented in this chapter.

2.2 Distributed SDN controllers

Fig. 2.1 shows a distributed SDN architecture with two controllers managing a single network divided in two domains. The traffic is exchanged directly among the controllers through the *east-west interface* [15], which is in addition to the *north-bound interface* (providing the APIs to interact with the controller at application level) and the *south-bound interface* (running a standard control protocol to manage the switching devices, as OpenFlow), both available in any SDN controller.

To understand the role of the traffic exchanged by the controllers, we start by describing an important result in the theory of distributed systems.

2.2.1 CAP theorem

Consistency of shared data in distributed systems is a well known and deeply investigated property. This property is achieved with quite complex protocols and algorithms [16]. The consistency dilemma is explained thoroughly using the famous CAP theorem [17] which states the impossibility of enjoying the following three

properties at the same time: Consistency, i.e. all the data reads access the latest written version of the data; Availability, i.e. all data is accessible and can be updated; Partition, i.e. the system is tolerant to node partitions.

Even if the proof of the CAP theorem is complex, a convincing scenario to understand this property is a storage system with the data replicated locally in two servers connected through a communication link. If availability and consistency are required at the same time (CA case), i.e. each server should be able to update the local data and access the most recent version of it, network partitions are not allowed, since the two servers must always be able to communicate an update to the other. Similarly, if availability and tolerance to partitions is required (AP case), i.e. each server should be able to update the local copy of the data, then consistency cannot be guaranteed anymore when partitions occur. Finally, if consistency and tolerance to partitions is required (CP case), i.e. the servers must access the most recent version of the data even in the case of partitions, availability cannot be guaranteed since each server cannot update the local copy in case of partitions. Depending on the pair of required guarantees (CA, AP or CP) in a distributed system, a large number of consistency protocols and algorithms have been devised and implemented so far.

2.2.2 Consistency in distributed SDN controllers

In a distributed SDN scenario, consistency means that all the controllers view the same network state, e.g., have the same local copy of the network topology and of the node/link availability state in their shared data structures. Any change of state occurring on each controller (due to, for example, new flow setups, link failures) must be promptly propagated to the other controllers according to one consistency protocol. If the controllers have an inconsistent view, the network policies may not run correctly and this can lead to potential network misbehaviors (as routing loops, packet drops, firewall leaks). For example, consider Fig. 2.1; if the communication between the east-west interfaces is not available, the control network is partitioned. In this case if there is a change in topology in controller B's domain, then it will not be propagated to controller A. Consequently controller A could take routing decisions based on an older view of the network topology in controller B's domain that could lead to unexpected behaviors.

In the theory of distributed systems, many consistency models have been defined. We concentrate here on just two of them, which have a direct application in SDN networks.

Eventual consistency model

This model provides a weak form of consistency, in sense that data modifications on a certain controller will be *eventually* propagated on all the other controllers. This implies that, for some time, some controllers may read values different from the actual updated ones; but after some time, all the controllers will have the updated values, given that they are able to communicate. This model is typically employed in distributed systems requiring high availability. The anti-entropy protocol, implemented in ONOS and described in Sec. 2.2.3, supports this consistency model.

Strong consistency model

This model ensures that each controller always reads the most updated version of the data. If certain data are not yet updated to all (or most of) the controllers, then they are not allowed to be read, thereby favoring consistency instead of availability. The RAFT consensus protocol, implemented in ONOS and described in Sec. 2.2.3, supports this consistency model.

The controllers exchange some control traffic, denoted as *inter-controller traffic*, through their east-west interfaces, to synchronize their shared data structures. The adopted consistency model heavily affects the inter-controller traffic, whose evaluation and modeling is the main contribution of our work.

2.2.3 Distributed ONOS

We now focus on the specific distributed architecture of ONOS controller, which allows to achieve a large scalability and availability, thanks to a distributed cluster of controllers. Each controller in the cluster is responsible of managing the switches under its domain, and updating their state on the distributed data stores. Each switch can connect to multiple ONOS controllers for reliability, but only one will be its *master* with full control on it in terms of read/write capabilities on the switch

forwarding tables. The other controllers are denoted as *slaves* and one of them takes the control of a switch whenever the master controller fails. Anytime a cluster of controllers is set up, each controller interacts with all the other controllers, thus the controllers are always logically connected in a full mesh according to a peer-to-peer approach, using a specific TCP port (9876) for their interaction. The controllers send and accept keep-alive messages to/from other controllers to monitor the other cluster members.

Two consistency protocols are implemented in v1.4.0 (Emu) and v1.8 (Ibis - Dec. 2016) versions of ONOS [18] to manage the distributed stores, each protocol tailored to guarantee a specific level of consistency.

Anti-Entropy Protocol

It is based on a simple gossip algorithm in which each controller chooses at random another controller in the cluster every 5 seconds and then sends a message, containing the timestamp of each entry, to compare the actual content of its store with the other one. After the synchronization messages are exchanged and the stores are updated based on the timestamp of each entry (i.e. more recent updates supersede the older ones), the two controllers become mutually consistent. This ensures that all the controllers achieve consensus according to an eventually consistent model. However, in parallel with the above scheme, whenever an update occurs in the store managed by a controller, this is immediately broadcasted to all the other controllers in the cluster.

RAFT Protocol

It is a recently proposed scheme [19] which provides strong consistency in ONOS. A RAFT implementation requires a cluster of nodes (i.e. controllers in our scenario) each having a database termed as the “log” which is replicated in all the nodes: each update is appended to this shared data structure. The consistency is coordinated by a leader node in the cluster, which is responsible for receiving update requests from all the other nodes and then relaying log updates to the other nodes. Once the majority of the followers have acknowledged the update, this is actually committed to the log. In the case of network partitions, only the side with the majority of the nodes is able to update the log, thus avoiding contemporary and conflicting updates in two

different network partitions. All the updates on the distributed stores are tracked using logical timestamps, which allow to reconcile conflicts based on the most recent updates.

In ONOS, multiple instances of RAFT protocol run simultaneously. The data structures in the distributed stores are partitioned into shards, where each shard is managed by a different RAFT instance. Partitioning is aimed at improving scalability. The total number of partitions is $N+1$ where N is the number of controllers in an ONOS cluster. The partitions are termed as $p_0, p_1, \ldots p_N$. Partition p_0 encompasses all the controllers in the cluster and is just for temporary storage, which is reset if the controller shuts down. For durable storage, the data is partitioned into N shards. The number of controllers that participate in each partition is $\min(3,N)$, i.e. each shard is shared among not more than 3 controllers. The partition p holding the value corresponding to a given key k within a data structure is chosen with a simple hash map $h(\cdot)$ as follows:

$$p = [h(k) \mod N] + 1 \tag{2.1}$$

Stores are the actual distributed data structures in ONOS. Each store is based on either the Anti-Entropy protocol, RAFT protocol or both. In particular, the main ONOS stores are:

- *Mastership store*, which keeps the mapping between each switch to its master. It is managed by RAFT protocol.
- *Network topology store*, which describes the network topology in terms of links and switches; consistency is achieved using the anti-entropy protocol.
- *Flow store*, which is responsible for backing flows of each switch from the master controller to the slave controller on detecting a change in the flow table. The details of the adopted consistency model are discussed in Sec. 2.5.
- *Host store*, which maintains the list of the network hosts. It is managed by RAFT protocol.
- *Application store*, which manages the inventory of applications, and adopts the anti-entropy protocol.
- *Intent store*, which manages the inventory of intents using the anti-entropy protocol. Intents are part of the ONOS Intent framework used by applications

to define which policy is operating on the network, without the details of how the data plane must be actually programmed.

- *Component configuration store*, which stores system-wide configurations for various software components in ONOS. It adopts the anti-entropy protocol.
- *Network configuration store*, which is used to store network configurations inserted into ONOS via the north-bound (e.g., REST API) or the south-bound API (e.g., OpenFlow). It adopts RAFT consensus algorithm.
- *Security mode store*, which manages permissions granted to applications using RAFT protocol. Instead, security violations are managed using the anti-entropy protocol.

Of all these distributed stores, the ones which are related to the data plane behavior are network topology store, flow store and host store, where each of them will be investigated in dedicated sections (Secs. 2.4-2.6). The other distributed stores are specific of each application and are neglected as part of the experimental work in order to keep our results general.

2.3 Methodology for inter-controller traffic analysis

For prototyping and testing, a test setup based on a standalone Ubuntu 14.10 server machine is used. A cluster of ONOS version 1.4.1 controllers runs in a set of Linux containers (LXC) [20] hosted on the server machine as shown in Fig. 2.2. LXC was chosen since containers are lighter on the CPU than virtual machines and do not show undesired background traffic, thus allowing to easily identify all the traffic generated by each instance of the controller. Notably, the adopted choice of the operating system (OS) virtualization is transparent for the controller instances, and thus our results hold for any other virtualization system compatible with the considered ONOS distribution.

We adopt Mininet-2.2.1 to emulate a network topology consisting of OpenFlow compliant software switches. Each switch is associated to one master controller and all the other slave controllers.

As shown in Fig. 2.2, three logical network topologies are created using virtual bridges available in Linux:

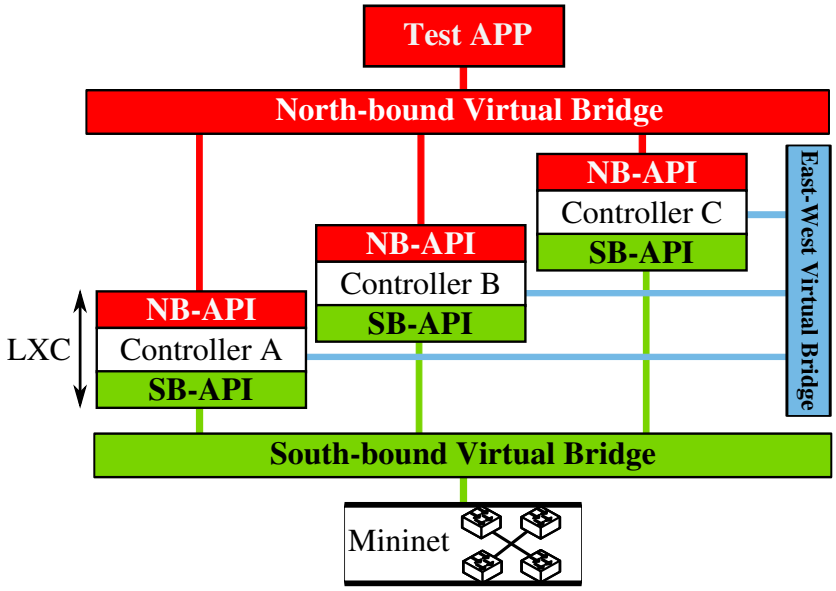


Fig. 2.2: Single-host testbed to investigate inter-controller traffic for distributed SDN controllers.

1. *North-bound Virtual Bridge*, connecting the controllers to our test application through the north-bound interfaces of the controllers;

2. *South-bound Virtual Bridge*, connecting the network emulated with Mininet to the controllers;

3. *East-West Virtual Bridge*, connecting directly the controllers to each other.

The use of separate virtual bridges simplifies traffic capture and management. We run Wireshark as a sniffer to capture the inter-controller traffic between any pair of controllers by capturing all the TCP traffic on the interface of a controller towards the other controller(s). ONOS uses port 9876 for the inter-controller communication, thus it is simple to identify such traffic. The total inter-controller traffic is sampled every $T_s = 0.1$ s to compute the throughput. The throughput samples are averaged through a sliding window of $T_w = 10$ s.

In each experiment, we start the controllers and then we wait until the initial transient phase ends. Here, the initial transient phase of the inter-controller is the initial exchange of all data stores between the controllers over the east-west interface until they are consistent with each other (which is marked by the observance of steady state traffic; more detail in Sec. 2.4). For our experiments, we wait for a

time of 120s, after starting the LXC containers hosting the ONOS controllers, to allow for the initial transient phase to end and reach a steady-state throughput as mentioned at the end of Sec. 2.3.1; where this steady state throughput is known as the *Zero Throughput* with further explanation in Sec. 2.4.1. Then, appropriate events are generated either on the south-bound interface through the terminal commands available in Mininet, or on the north-bound interface using our test application, which leverages the APIs exposed by the controller. To repeat an experiment, we restart by rebooting the LXC container for each SDN controller, to avoid residual data due to previous experiments. The reboot procedure is necessary since affecting the "tombstone" inter-controller traffic, as explained further in Sec. 2.4.

2.3.1 Implementation approaches for consistency models

The specific implementation of the shared data structures across the controllers and the adopted consistency model have significant impact on the inter-controller traffic. We categorize the inter-controller traffic as a combination of the following types of updates:

- *incremental updates, or full updates*: this feature describes the actual information that is exchanged among controllers. In the case of incremental updates, only the differences with respect to the previous updates are exchanged. Since incremental updates must rely on a coherent update of the previous states, the approach is typically employed by a strong consistency model. Instead, in the case of full updates, the whole data structure is exchanged. Full updates are typically exchanged for eventually consistent data structures, due to the unreliable state coherence among data structures.
- *periodic updates, or event-driven updates*: this feature describes when the updates are issued. Periodic updates are generated periodically over the time, whereas event-driven updates are triggered by specific change of states or events.

All the four combinations of the two above features are possible in practice, as shown later in Secs. 2.4-2.6.

The overall inter-controller traffic is due to the superposition of the synchronization of different data structures, each of them with a specific feature. Thus, to

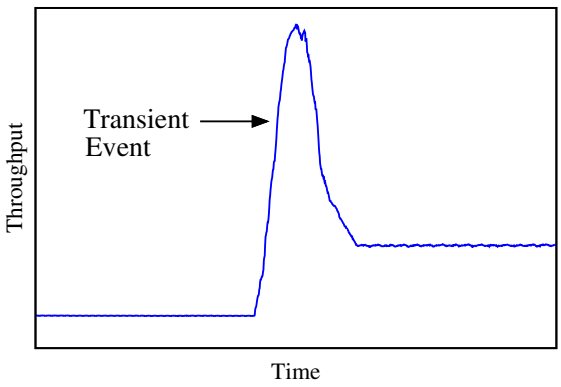


Fig. 2.3: Transient phase detection

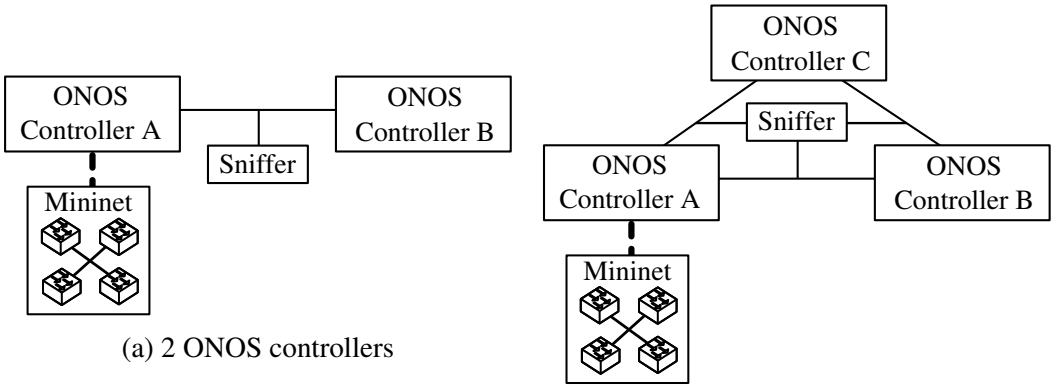
understand the traffic due to a specific data structure, we specifically modify the data on just a single data structure, generating carefully crafted events in the test application or in the Mininet topology.

We measure the traffic due to each update event in terms of amount of data or throughput. For the first case, we measure the additional traffic generated during the transient phase. For the second case, we just evaluate the derivative of the cumulative amount of exchanged traffic. Notably, the transient phase is identified as included between two periods of steady-state throughput values, as shown in Fig. 2.3. Interestingly, as shown in Sec. 2.4, the throughput after the transient phase may be different from the initial one.

2.4 Distributed Topology Store

We evaluate the traffic exchanged among the ONOS controllers due to the network topology store. Our results are general since we expect them to hold for a broad set of topologies with arbitrary partition of the network into controller domains. To highlight the role of the topology, we adopt a time-variant topology in which the number of active switches and active edges changes with the time. By measuring the variation of the inter-controller traffic, we are able to understand the detailed effect of modifications in the topology store.

We consider two main scenarios, both shown in Fig. 2.4: the first one with 2 controllers and the second one with 3 controllers belonging to the same cluster. We



(a) 2 ONOS controllers

(b) 3 ONOS controllers

Fig. 2.4: Experimental testbed

denote with A, B and C the instances of the controller, running within the same controller cluster. Let S be the total number of switches in the network topology and L be the corresponding number of (bidirectional) links. We adopt some simple test network topologies to highlight the individual contribution of each network element (switch or link) and thus obtain general results holding for a broad set of topologies. In the *isolated topology* we have S isolated switches without links ($L = 0$). In the *linear topology*, S switches are connected linearly ($L = S - 1$) as shown in Fig. 2.5. In the *star topology*, $S - 1$ switches are connected to the same central switch ($L = S - 1$) as shown in Fig. 2.6. We repeat all the experiments 20 times and compute the 98% confidence intervals.

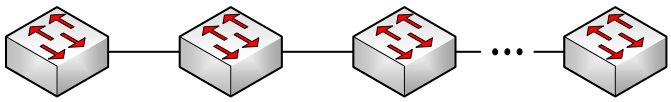


Fig. 2.5: Linear topology

The inter-controller traffic generated among the controllers due to the topology store is *periodic with full updates*. This is due to the adoption of anti-entropy protocol for maintaining consistency in network topology store. Another contribution is *periodic with incremental updates* and it is due to the Link Layer Discovery Protocol (LLDP) packets sent for the topology discovery received on the south-bound interface. LLDP is used by the SDN controller to discover links in the network topology. The SDN controller periodically sends LLDP packets using OpenFlow *Packet_Out* messages to the OpenFlow devices registered with the controller. The devices then flood the LLDP through all its ports. The adjacent devices receive the LLDP packet

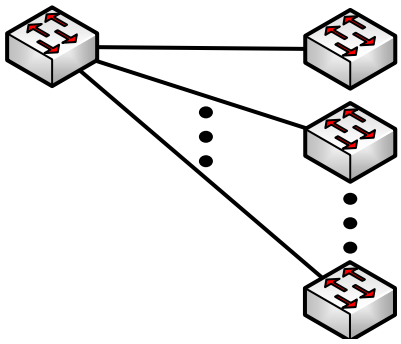


Fig. 2.6: Star topology

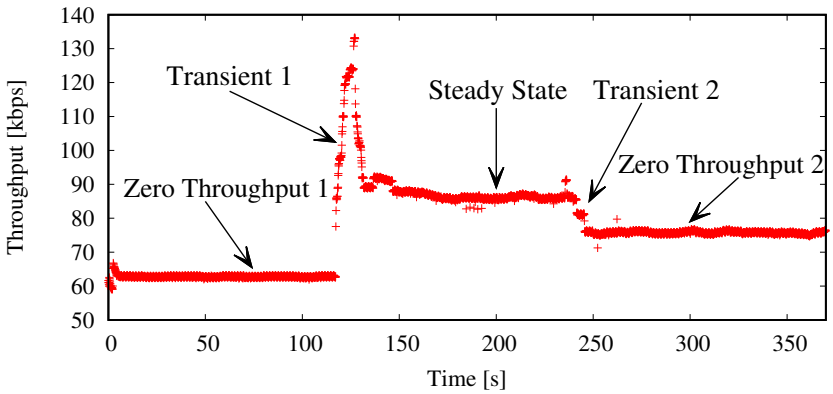


Fig. 2.7: Traffic from controller A to B by adding and removing a linear topology $S = 4$, $L = 3$

and forward it to the SDN controller via an OpenFlow *Packet_In* message. The controller upon receiving the OpenFlow *Packet_In* can discover the link by checking the contents of the received message.

2.4.1 Transient behavior in the linear topology with 2 controllers

In Fig. 2.7 we show the communication throughput from controller A to controller B for a linear topology with $S = 4$, with all the switches managed by controller A. We start with no topology ($S = 0$, $L = 0$) added to controller A; at time 120s the linear topology is added ($S = 4$, $L = 3$); at time 240s the linear topology is removed. At the beginning of the experiment we observe an initial communication of 63 kbps (denoted as *Zero Throughput 1*). When the linear topology is added, after a short transient phase, the traffic reaches 88 kbps. When the network is removed,

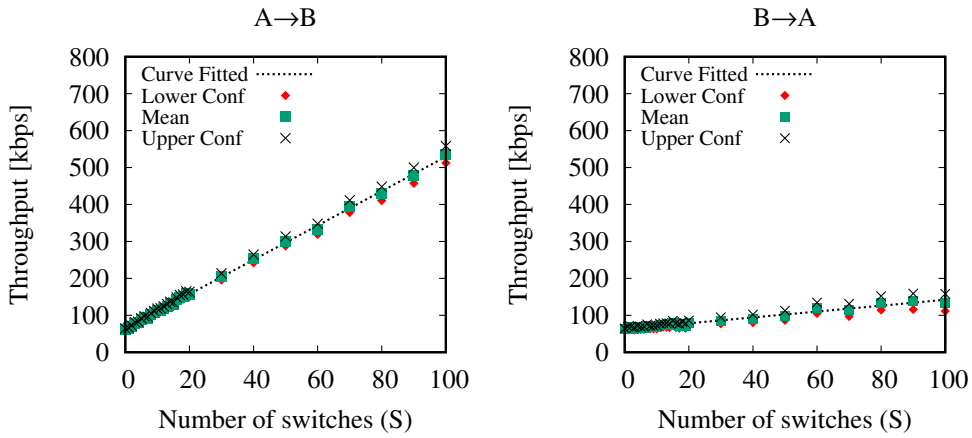


Fig. 2.8: Intercontroller traffic, where isolated topology is associated to controller A (master) in the scenario with 2 controllers

the throughput reaches 78 kbps (denoted as *Zero Throughput 2*). This value is different from the initial one at the beginning of the experiment, and it is due to the exchanged "tombstone" traffic. Tombstone traffic is due to the anti-entropy protocol and refer to devices, links and hosts which have been removed from the active topology. The reason for it is to react faster to network partitions. Indeed, in the case of temporary network partitions, keeping tombstones minimizes the variation in the internal topology store, and thus the allocation/deallocation of memory for the internal data structures. Notably, after each experiment, the LXC container is rebooted so that no tombstone traffic persists in the observed traffic.

2.4.2 Scenario with 2 controllers

We investigate the traffic exchanged by controllers A and B in steady state for different sizes of the topology, in which all the switches are under A's control. Fig. 2.8 shows the throughput from A $\rightarrow$ B and vice versa, when an isolated topology is added to controller A. We show also the confidence intervals and one linear curve fitting the experimental measurements. Similarly, Fig. 2.9 shows the throughput when a linear topology is added to controller A. Both graphs show that the throughput is increasing linearly in both communication directions. This is coherent with the linear growth of the internal data structures, based on hash tables. Moreover, the throughput for A $\rightarrow$ B is larger than B $\rightarrow$ A. If we consider that the topology store is distributed with the anti-entropy protocol, we should expect a symmetric behavior. Instead at

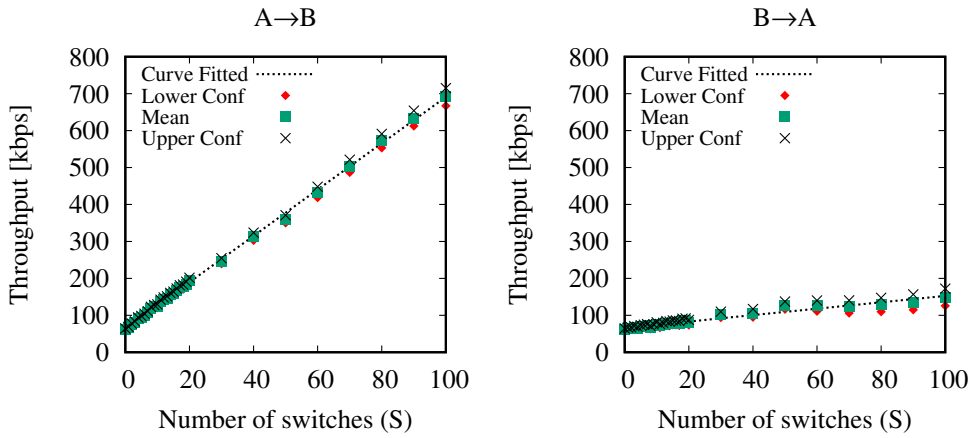


Fig. 2.9: Intercontroller traffic, where linear topology is associated to controller A (master) in the scenario with 2 controllers

controller A, the topology is periodically refreshed (even if not changing) through the LLDP packets received on the south-bound interface for topology discovery. This causes an update on the topology store, which generates additional traffic from A to B and causes the asymmetry. In addition, port and flow statistics gathered periodically by controller A are also sent to B.

Due to the internal data structures, whose memory occupancy grows linearly with the number of elements (nodes and links), we can assume that the exchanged traffic B in each direction is proportional to the size of the topology store:

$$B = S \cdot b^s + L \cdot b^l + b^0 \tag{2.2}$$

where we used the notation in Table 2.1. By applying (2.2) to the linear topology ($L = S - 1$) and to the isolated topology ($L = 0$) considered in our experiments, we can write the following system of equations, assuming that A is master controller of all the switches in the network; and where B^I and B^L are the throughputs in case of isolated and linear topologies respectively:

$$\begin{cases} B^L_{\mathrm{A}\to\mathrm{B}} = S \cdot b^s_{\mathrm{A}\to\mathrm{B}} + (S-1) \cdot b^l_{\mathrm{A}\to\mathrm{B}} + b^0 \\ B^L_{\mathrm{B}\to\mathrm{A}} = S \cdot b^s_{\mathrm{B}\to\mathrm{A}} + (S-1) \cdot b^l_{\mathrm{B}\to\mathrm{A}} + b^0 \\ B^I_{\mathrm{A}\to\mathrm{B}} = S \cdot b^s_{\mathrm{A}\to\mathrm{B}} + b^0 \\ B^I_{\mathrm{B}\to\mathrm{A}} = S \cdot b^s_{\mathrm{B}\to\mathrm{A}} + b^0 \end{cases} \tag{2.3}$$

Table 2.1: Notation for traffic in the scenario with 2 controllers x and y

Symbol	Meaning
B	generic unidirectional throughput
b^0	generic zero throughput
b^s	average unidirectional throughput per switch
b^l	average unidirectional throughput per intra-domain link
b^d	average unidirectional throughput per inter-domain link, (shared also by target controller for 3 controller scenario)
b^e	average unidirectional throughput per inter-domain link, (external to target controller for 3 controller scenario)
$B^I_{x\to y}$	throughput from x to y in isolated topology
$B^L_{x\to y}$	throughput from x to y in linear topology
$B^\star_{x\to y}$	throughput from x to y in star topology
$b^s_{x\to y}$	average throughput from x to y per switch
$b^l_{x\to y}$	average throughput from x to y per intra-domain link

This system can be solved by measuring $B^L_{x\to y}$, $B^I_{x\to y}$, for any $x, y \in \{\mathrm{A}, \mathrm{B}\}$ $(x \neq y)$ and b^0 and thus estimating the remaining unknown values of per-link and per-switch throughput.

Based on our measurements, we observe always a constant value of zero throughput equal to $b^0 = 62.46$ kbps (obtained with 3.6% accuracy at 96% confidence level) for the both directions, given our measurements. Thus, solving the system in (2.3), we obtain experimentally:

$$b^l_{\mathrm{A}\to\mathrm{B}} = 1.63 \text{ kbps} \qquad b^l_{\mathrm{B}\to\mathrm{A}} = 0.11 \text{ kbps} \tag{2.4}$$

$$b^s_{\mathrm{A}\to\mathrm{B}} = 4.65 \text{ kbps} \qquad b^s_{\mathrm{B}\to\mathrm{A}} = 0.80 \text{ kbps} \tag{2.5}$$

So far all the network switches have been associated to the same controller. In order to extend our model to a broad set of topologies, arbitrarily partitioned among two controller domains, we need to evaluate the effect of inter-domain links, i.e. connecting one switch in one domain with another in the other domain. We consider the star topology in Fig. 2.10, in which we vary the number of switches and consequently the number of inter-domain links. Now the observed throughput in one direction is obtained by summing the following contributions: B^I for 1 switch to model the switch in controller A's domain; B^I for $S-1$ switches to model the $S-1$

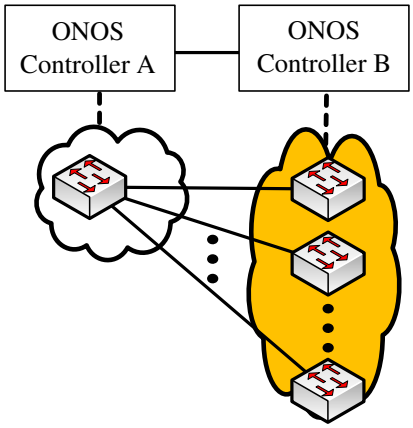


Fig. 2.10: Scenario with a star topology

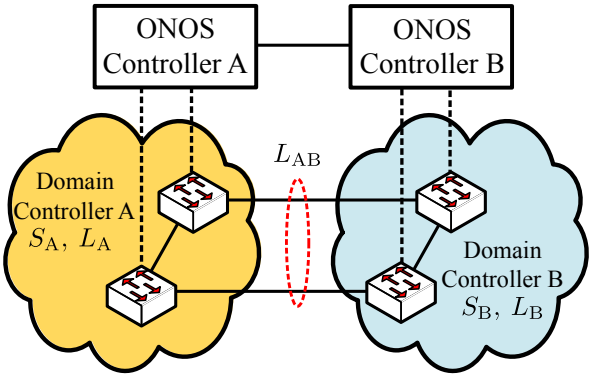


Fig. 2.11: Notation depicting the network topology in the scenario with 2 controllers

switches in B's domain; $S-1$ times the average throughput per inter-domain link b^d as shown in (2.6) using the values from (2.4)-(2.5).

$$\begin{cases} B^{\star}_{\mathrm{A}\to\mathrm{B}} = 1\cdot 4.65 + (S-1)\cdot 0.80 + (S-1)b^d \\ B^{\star}_{\mathrm{B}\to\mathrm{A}} = (S-1)\cdot 4.65 + 1\cdot 0.80 + (S-1)b^d \end{cases} \tag{2.6}$$

Using the same methodology before and exploiting the estimated values obtained so far, we estimate that the average throughput per inter-domain link is

$$b^d = 0.63 \text{ kbps} \tag{2.7}$$

By combining the results so far and the estimated throughput in Equations (2.4) to (2.7), we can claim the following, by referring to the notation in Table 2.2:

Property 1 *In an arbitrary network managed by a ONOS cluster of 2 controllers A and B, the traffic exchanged from controller x to controller y is:*

$$B_{x\to y} = 62.46 + 4.65 \cdot S_x + 1.63 \cdot L_x + 0.80 \cdot S_y + 0.11 \cdot L_y + 0.63 \cdot L_{xy} \ \ [kbps] \quad (2.8)$$

for $x = A$ and $y = B$, or for $x = B$ and $y = A$.

Let $B_{TOT} = B_{\mathrm{A}\to\mathrm{B}} + B_{\mathrm{B}\to\mathrm{A}}$ be the total exchanged traffic among the two controllers. Referring to the notation in Table 2.2 and depicted in Fig. 2.11, we compute:

$$\begin{aligned} B_{TOT} &= B_{\mathrm{A}\to\mathrm{B}} + B_{\mathrm{B}\to\mathrm{A}} \\ &= 62.46 + 4.65 \cdot S_{\mathrm{A}} + 1.63 \cdot L_{\mathrm{A}} + 0.80 \cdot S_{\mathrm{B}} + 0.11 \cdot L_{\mathrm{B}} + 0.63 \cdot L_{\mathrm{AB}} \\ &\quad +62.46 + 4.65 \cdot S_{\mathrm{B}} + 1.63 \cdot L_{\mathrm{B}} + 0.80 \cdot S_{\mathrm{A}} + 0.11 \cdot L_{\mathrm{A}} + 0.63 \cdot L_{\mathrm{AB}} \\ &= 124.92 + 5.45(S_{\mathrm{A}} + S_{\mathrm{B}}) + 1.74(L_{\mathrm{A}} + L_{\mathrm{B}}) + 1.26 \cdot L_{\mathrm{AB}} \\ &\quad +1.74 \cdot L_{\mathrm{AB}} - 1.74 \cdot L_{\mathrm{AB}} \\ &= 124.92 + 5.45(S_{\mathrm{A}} + S_{\mathrm{B}}) + 1.74(L_{\mathrm{A}} + L_{\mathrm{B}} + L_{\mathrm{AB}}) \\ &\quad -0.48 \cdot L_{\mathrm{AB}} \end{aligned} \quad (2.9)$$

Using the notation from Table 2.2 in (2.9), we claim:

Corollary 1 *In an arbitrary network managed by a ONOS cluster of 2 controllers A and B, it holds*

$$B_{TOT} = 124.92 + 5.45 \cdot S + 1.74 \cdot L - 0.48 \cdot L_{AB} \ \ [kbps]$$

Thus, the total inter-controller traffic grows linearly with respect to the number of switches and links in the topology.

We validated the above formulas by considering multiple scenarios, including full mesh topologies, ring topologies, irregular topologies. All the experimental results have been always compatible with the model prediction of Property 1 within 98% confidence interval.

Table 2.2: Notation describing the network topology in the scenario with 2 controllers. Let x and y be the 2 controllers, with $x, y \in \{\mathrm{A}, \mathrm{B}\}$.

Symbol	Meaning
S_x	number of switches in controller x's domain
$S = S_\mathrm{x} + S_\mathrm{y}$	total number of switches in the network
L_x	number of intra-domain links in controller x's domain
L_xy	number of inter-domain links
$L = L_\mathrm{x} + L_\mathrm{y} + L_\mathrm{xy}$	total number of links in the network

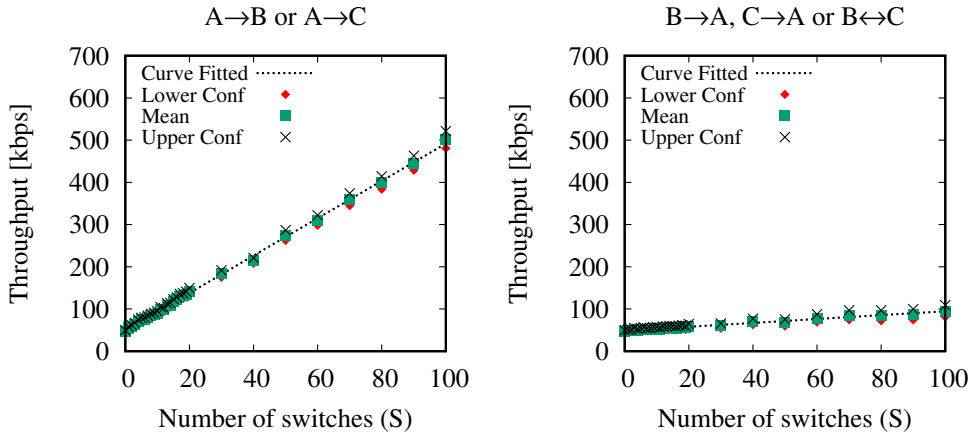


Fig. 2.12: Intercontroller traffic when isolated topology is added to controller A in the scenario with 3 controllers

2.4.3 Scenario with 3 controllers

The methodology adopted in the previous scenario is now extended to the 3 controllers scenario using the configuration shown in Fig. 2.4. We start by adding the topology to controller A. For symmetry, the throughput $B_{\mathrm{A}\to\mathrm{B}} = B_{\mathrm{A}\to\mathrm{C}}$; as no topology is added to controllers B and C, similarly $B_{\mathrm{B}\to\mathrm{A}} = B_{\mathrm{B}\to\mathrm{C}} = B_{\mathrm{C}\to\mathrm{A}} = B_{\mathrm{C}\to\mathrm{B}}$. Fig. 2.12 shows the throughput from A to B and vice versa, when an isolated topology is added to controller A.

Fig. 2.13 shows the throughput when a linear topology is added. As compared to Fig. 2.8 and 2.9, the throughput values in the 3-controller case are lower than the 2-controller case. This is due to the anti-entropy protocol: periodically, each controller randomly selects another controller to synchronize the network topology. Say the synchronization rate for each controller is λ. Thus the average contribution of this process on each link is $3\lambda/6 = \lambda/2$, since 6 possible links are present with

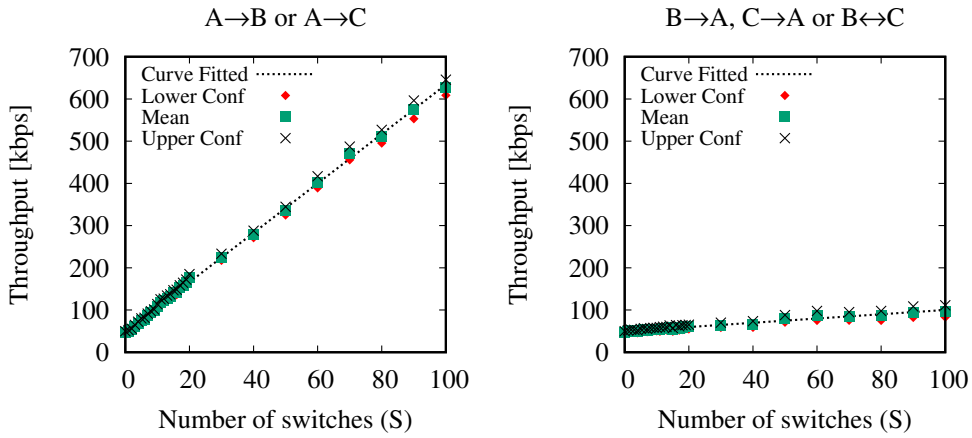


Fig. 2.13: Intercontroller traffic when linear topology is added to controller A in the scenario with 3 controllers

3 controllers. Instead, in the case of 2 controllers, the average contribution was $2\lambda/2 = \lambda$. Thus, a reduction of a factor 2 in the throughput due to the enti-entropy is expected. In the figure, the reduction is much lower, due to the lower impact of this protocol with respect to the topology updates sent by the master controller of a switch and caused by LLDP packets, as explained in Sec. 2.4.2. Globally, the throughput in each direction is still proportional to the size of the topology store. Following the assumption while deriving (2.2) and considering that the topology is only added to controller A, the following system of equations can be written with the notation in Table 2.1:

$$\begin{cases} B^L_{\mathrm{A}\to\mathrm{B}} = S \cdot b^s_{\mathrm{A}\to\mathrm{B}} + (S-1) \cdot b^l_{\mathrm{A}\to\mathrm{B}} + b^0 \\ B^L_{\mathrm{B}\to\mathrm{A}} = S \cdot b^s_{\mathrm{B}\to\mathrm{A}} + (S-1) \cdot b^l_{\mathrm{B}\to\mathrm{A}} + b^0 \\ B^I_{\mathrm{A}\to\mathrm{B}} = S \cdot b^s_{\mathrm{A}\to\mathrm{B}} + b^0 \\ B^I_{\mathrm{B}\to\mathrm{A}} = S \cdot b^s_{\mathrm{B}\to\mathrm{A}} + b^0 \end{cases}$$

which can be solved numerically. The zero throughput between any two controllers here is $b^0 = 47.81$ kbps (obtained with 4.15% accuracy at 98% confidence level).

Thus, we obtain:

$$b^l_{\mathrm{A}\to\mathrm{B}} = b^l_{\mathrm{A}\to\mathrm{C}} = 1.43 \text{ kbps} \tag{2.10}$$

$$b^l_{\mathrm{B}\to\mathrm{A}} = b^l_{\mathrm{B}\to\mathrm{C}} = b^l_{\mathrm{C}\to\mathrm{A}} = b^l_{\mathrm{C}\to\mathrm{B}} = 0.06 \text{ kbps} \tag{2.11}$$

$$b^s_{\mathrm{A}\to\mathrm{B}} = b^s_{\mathrm{A}\to\mathrm{C}} = 4.43 \text{ kbps} \tag{2.12}$$

$$b^s_{\mathrm{B}\to\mathrm{A}} = b^s_{\mathrm{B}\to\mathrm{C}} = b^s_{\mathrm{C}\to\mathrm{A}} = b^s_{\mathrm{C}\to\mathrm{B}} = 0.46 \text{ kbps} \tag{2.13}$$

To extend our empirical model to a broad set of topologies, arbitrarily partitioned among the two controller domains, the star topology in Fig. 2.10 is considered albeit with 3 controllers and no switch added to controller C, in which we vary the number of switches and thus the inter-domain links. In this scenario, the amount of throughput originating from each controller to the other two controllers is different, since different number of switches are added to each controller. Hence, here $B_{\mathrm{B}\to\mathrm{A}} = B_{\mathrm{B}\to\mathrm{C}} \neq B_{\mathrm{C}\to\mathrm{A}} = B_{\mathrm{C}\to\mathrm{B}}$. Furthermore, the average unidirectional throughput per inter-domain link in this case is b^d for controllers A and B, but it is b^e for controller C, since the links are external to it but of inter-domain type. Now the observed throughput in one direction is obtained by summing the following contributions: B^I for 1 switch to model the switch in A's domain; B^I for $S-1$ switches to model the $S-1$ switch in B's domain; $S-1$ times the average throughput per inter-domain link b^d and $S-1$ times the average throughput per external inter-domain link b^e.

Using the same methodology before, and exploiting the estimated values obtained so far, we are able to estimate that the average throughput per inter-domain link as:

$$b^d = 0.77 \text{ kbps} \quad b^e = 0.15 \text{ kbps} \tag{2.14}$$

By combining the results so far and the estimated throughput in Equations (2.10) to (2.14), we can claim the following:

Property 2 *In an arbitrary network managed by a ONOS cluster of 3 controllers A, B and C, the traffic exchanged from controller x to controller y is:*

$$\begin{aligned} B_{x\to y} = 47.81 + 4.43 \cdot S_x + 1.43 \cdot L_x + 0.46 \cdot (S_y + S_z) + \\ 0.06 \cdot (L_y + L_z) + 0.77 \cdot (L_{xy} + L_{xz}) + \\ 0.15 \cdot L_{yz} \ \ [kbps] \end{aligned} \tag{2.15}$$

Table 2.3: Notation describing the network topology in the scenario with 3 controllers. Let x, y be two distinct controllers, with $x, y \in \{\text{A,B,C}\}$.

Symbol	Meaning
S_x	number of switches in controller x's domain
$S = S_\text{A} + S_\text{B} + S_\text{C}$	total number of switches in the network
L_x	number of intra-domain links in controller x's domain
L_{xy}	number of inter domain links between x and y
$L = L_\text{A} + L_\text{B} + L_\text{C}$ $+L_\text{AB} + L_\text{BC} + L_\text{AC}$	total number of links in the network
L_{ID}	total number of inter-domain links in the network

for any selection of distinct controllers $x, y, z \in \{A,B,C\}$ (i.e. such that $x \neq y$, $x \neq z$ and $y \neq z$).

Let $B_{TOT} = B_{\text{A}\to\text{B}} + B_{\text{A}\to\text{C}} + B_{\text{B}\to\text{A}} + B_{\text{B}\to\text{C}} + B_{\text{C}\to\text{A}} + B_{\text{C}\to\text{B}}$ be the total exchanged traffic among the 3 controllers. Referring to the notation in Table 2.3 and using the similar methodology used to derive (2.9), we can claim:

Corollary 2 *In an arbitrary network managed by a ONOS cluster of 3 controllers, the total traffic exchanged among the 3 controllers is*

$$B_{TOT} = 286.86 + 10.7 \cdot S + 3.10 \cdot L + 0.28 \cdot L_{ID} \ \ [kbps]$$

where $L_{ID} = L_\text{AB} + L_\text{BC} + L_\text{AC}$. Thus, also in this scenario, the total traffic appears to be proportional to the number of switches and the number of edges in the topology.

2.4.4 Inter-controller traffic in real ISP topologies

To prove the wide applicability of our approach, we apply the empirical models of Property 1 (for 2 controllers) and Property 2 (for 3 controllers) to 262 real ISP network topologies obtained from the Internet Topology Zoo [21] to obtain the inter-controller traffic in case of a distributed ONOS cluster managing the whole ISP network. The number of nodes and edges in each ISP is shown in Fig. 2.14 and show a high variety, even if in most of the cases the topology graph is not dense. This is reasonable, since for a large (in term of geographical distance) ISP, dense graphs are expensive and this fact advocates a careful design of the in-band communication network to support inter-controller traffic.

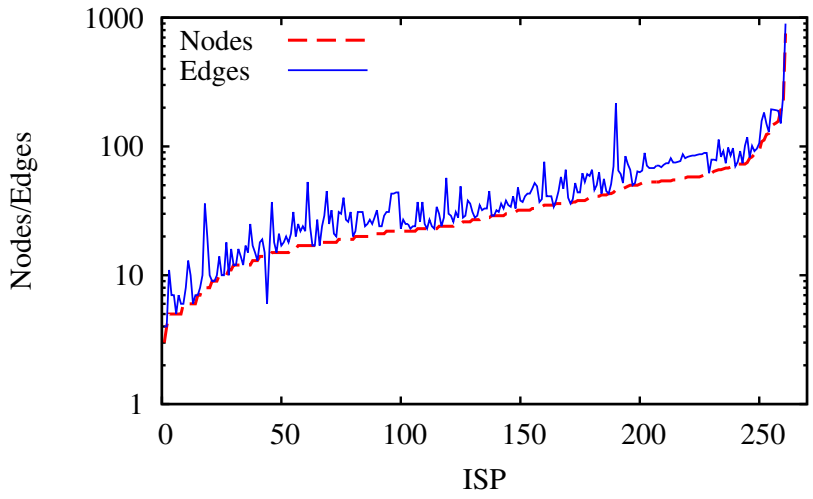


Fig. 2.14: Size of the network topologies considered for the inter-controller traffic in real ISP topologies

In order to obtain results regarding the inter-controller traffic that are independent from the controller chosen as the master for each switch, we evaluate the maximum and minimum value of the inter-controller traffic by assuming (without loss of generality) that each switch in the ISP topology is connected to a single controller denoted as A. This is because the contribution to the inter-controller traffic for a controller is maximum when all the switches are in controller A's domain (i.e. $S_{\mathrm{A}} = S$, $L_{\mathrm{A}} = L$, $S_{\mathrm{B}} = L_{\mathrm{B}} = 0$), as when applying (2.8), the coefficients for S_{A} and L_{A} are larger. Conversely, this assumption minimizes the traffic generated by controller B towards A. A similar argument can be used in (2.15) to show that the upper and lower bounds can still be obtained by associating all the switches to controller A.

Fig. 2.15 shows the average amount of inter-controller traffic for each ISP, exchanged between pairs of controllers. In the case of 2 controllers, traffic A $\rightarrow$ B and B $\rightarrow$ A give the maximum and minimum values. In the case of 3 controllers, traffic A $\rightarrow$ B or A $\rightarrow$ C provide the maximum values, whereas B $\rightarrow$ A, C $\rightarrow$ A or B $\leftrightarrow$ C provide the minimum. According to our experiments, as an example, the maximum inter-controller traffic in the 2-controllers scenario for the 261$^{\mathrm{st}}$ ISP is $B_{\mathrm{A}\rightarrow\mathrm{B}} = 5029$ kbps and the minimum is $B_{\mathrm{B}\rightarrow\mathrm{A}} = 763.75$ kbps. Both values are practically relevant, since for a generic partitioning of the network in two controller domains, a bandwidth of about 1-10 Mbps must be guaranteed among the pair of controllers, just to synchronise the topology store.

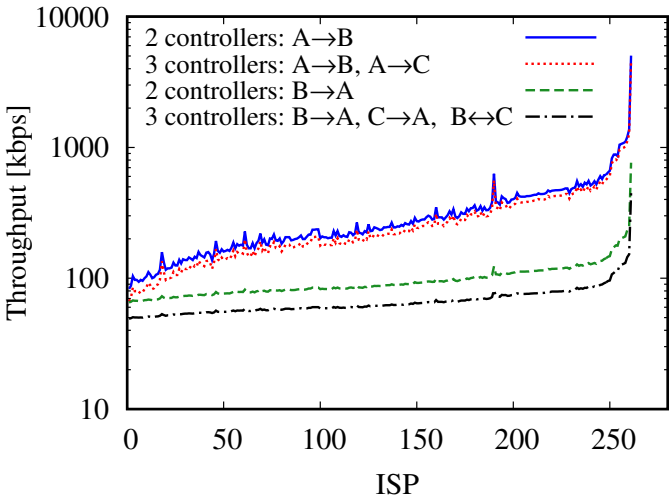


Fig. 2.15: Estimated inter-controller traffic, between pairs of controllers, due to the topology store for realistic ISPs

Similarly in the 3-controller case, as an example, the maximum inter-controller traffic for the 261st ISP is $B_{A\to B} = 4668.8$ kbps and the minimum is $B_{B\to A} = 447.73$ kbps. Also in this case the actual traffic is practically relevant, since about 1-10 Mbps is again required to support the communication among any pair of controllers. By comparing the results in Fig. 2.15 referred to different number of controllers, the traffic reduction is evident for 3 controllers with respect to 2 controllers case, as already observed in Sec. 2.4.3.

2.5 Distributed Flow Store

In ONOS, a copy of each switch's flow table is maintained in the flow store by its respective master controller and by the first slave controller, i.e. the new master if the current master fails. We investigate the impact of modifying the switches' flow tables by OpenFlow `flow-mod` commands.

To synchronize the flow stores within the ONOS cluster, according to the code available in [22], the following process occurs: every 2 seconds, the master controller checks for any change in the flow table of each switch under its domain since the last backup to the slave controller. The change detection is based on comparing the time when the flows changed in a switch and the time of the last backup to the slave

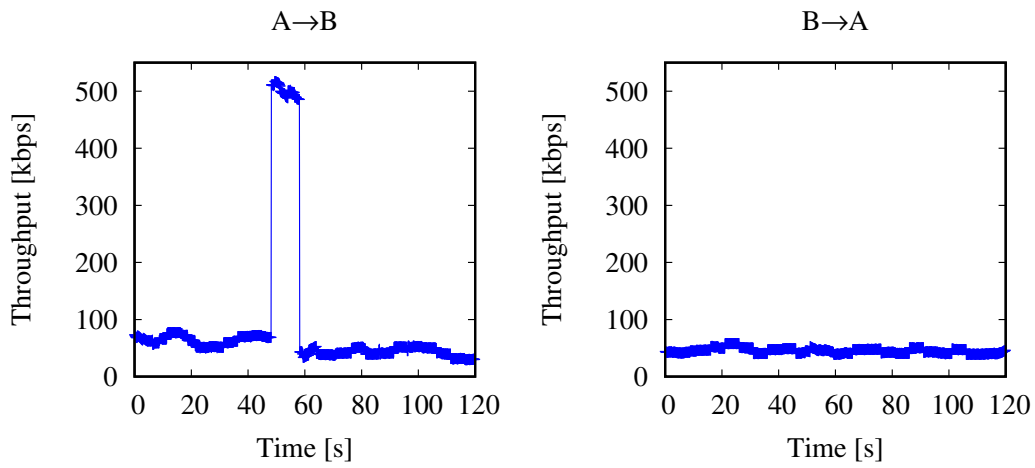


Fig. 2.16: Inter-controller traffic, depicting the effect of a single flow addition the occurring at time 50 s on the synchronization of the flow store

controller. Interestingly, in the case of a single modification of a flow, the *whole* flow store of the corresponding switch is copied to the slave controller.

As an example, consider the throughput measurement shown in Fig. 2.16, referring to the scenario in which a switch is connected to master controller A and slave controller B. Initially, the switch has 5004 flows installed in it. At around time 50s, one additional flow is installed on the switch, which causes the controller A to backup the whole flow table (5005 flows) to controller B. This results in a transient traffic increase, just for the traffic A $\rightarrow$ B.

2.5.1 Experimental methodology

We describe here the adopted methodology to calculate data exchanged per flow, i.e. taking into account the contribution of each individual flow. According to the previous observations, the flow store backup from the master controller to one slave controller is a transient phenomenon, thus the inter-controller traffic is *event-driven with full updates*, according to the classification in Sec. 2.3.1. By observing the traffic with the sniffer, we discover that ONOS adds the string “flow-backup” in its packets when backing the switch flow table. Thanks to this observation, we can easily isolate the traffic due to the flow table backup.

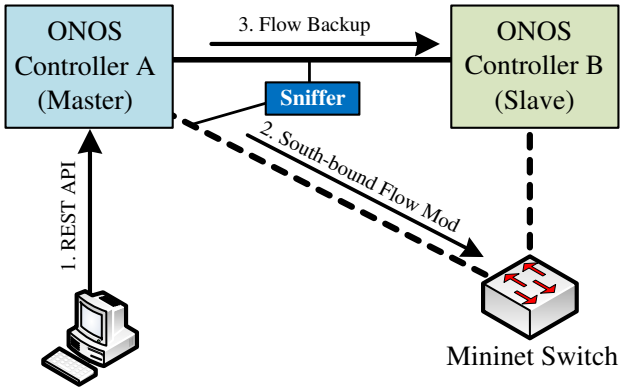


Fig. 2.17: Methodology to investigate the effect of flow modifications in the inter-controller traffic

We adopt the testbed illustrated in Fig. 2.17. The topology consists of one isolated switch connected to one master and one slave controller. The flows to be installed are configured on the master controller through the ONOS north-bound REST APIs (step 1). As a consequence, the flows are installed on the switch via the controller south-bound interface using OpenFlow (step 2) and then the controller backups the flow table to the slave controller (step 3).

To obtain general results, we test different types of flows with different versions of OpenFlow, while varying the number of flows, in order to evaluate the minimum and the maximum amount of data exchanged per flow. Let F be number of new added flows whose effect must be analyzed. We start by installing $F-1$ flows. After the traffic has reach a steady state, we install a single additional flow, in order to avoid multiple backups. Thanks to the traffic trace, we calculate the data exchanged per flow by computing the ratio of observed data on the network (in terms of Ethernet packet size) by the number of flows in the table.

The amount of per-flow data depends on the adopted "match" and "actions" fields adopted in the experiments, which in turn, depend also on the specific version of OpenFlow. ONOS version 1.4.1 supports two versions of OpenFlow: 1.0 [23] and 1.3 [24]. In order to get general results, we devise two types of flow definitions to be added in the table. *Type-1* is forged to be the flow definition with the minimum size, corresponding to the smallest `flow-mod` packet (on the south-bound interface) and thus the minimum inter-controller traffic. *Type-2* is instead forged to be the flow definition with the maximum size. In order to induce a constant synchronization traffic among the controllers, we generate a new flow by changing the value of just

Table 2.4: Types used for OpenFlow 1.0 experiments

(a) Type-1

	Field
Match	EtherType
Action	Output to Controller

(b) Type-2

	Field
Match	Input Port Ethernet Source/Destination Ethernet Type IPv4 Source / Destination IP Protocol Type / DSCP TCP Source/Destination Port
Action	Output to Controller Change VLAN ID / PCP POP VLAN Change Ethernet Source / Destination Change IPv4 Source / Destination

one matching field. For OpenFlow 1.0, we configure Type-1 flow definitions by just setting the EtherType matching field and a basic forward action, as shown in Table 2.4a, and we vary just the EtherType field for each new flow. Instead, we configure Type-2 flow definitions by setting all the 10 matching fields available in the REST APIs exposed by ONOS for OpenFlow 1.0 and all the 8 allowed actions, as shown in Table 2.4b. Similarly, for OpenFlow 1.3, we use the same Type-1 definitions as before, shown in Table 2.5a. Instead, for Type-2 definitions we set all 16 match fields and all the 11 available actions in Table 2.5b. Notably, we exploit IPv6 fields since they require a larger number of bits for their definitions. In all Type-2 definitions, we vary the TCP source port field for each new flow.

2.5.2 Experimental results

Fig. 2.18 shows the amount of data exchanged per flow averaged on 10 experiments, for all the 4 possible cases, combining Type-1 and Type-2 flow definitions with the two considered OpenFlow versions. As a reminder, this data includes all the packet overheads starting from the Ethernet PDU. All the graphs show that per-flow data converge to a fixed value, which can be evaluated for a large enough number of flows. Table 2.6 compares the inter-controller data for each flow, evaluated for the fixed value obtained in Fig. 2.18 with the value obtained by observing the size of the

Table 2.5: Types used for OpenFlow 1.3 experiments

(a) Type-1

	Field
Match	EtherType
Action	Output to Controller

(b) Type-2

	Field
Match	Input Port Metadata Tunnel ID VLAN ID / PCP Ethernet Source / Destination / Type IPv6 Source / Destination / Flow label IP Protocol Type / DSCP / ECN TCP Source / Destination
Action	Output to Controller Change VLAN ID / PCP POP VLAN Change Ethernet Source / Destination Change Tunnel ID Change IPv6 Source / Destination Change TCP Source / Destination

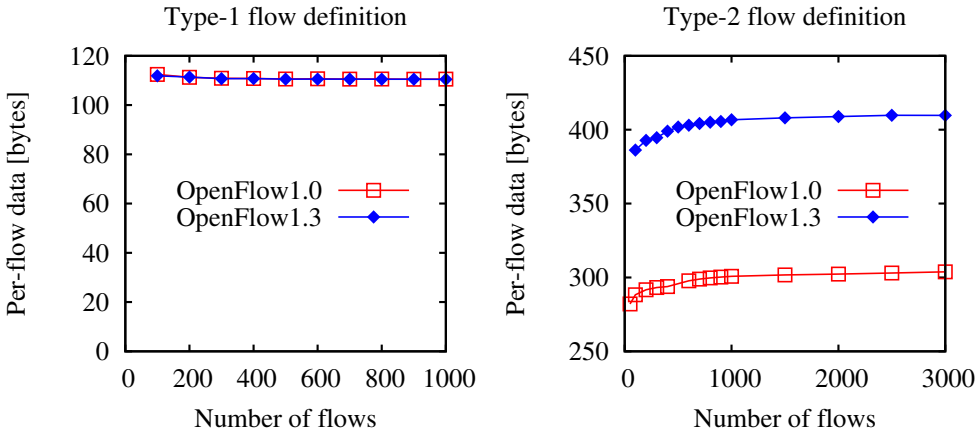


Fig. 2.18: Inter-controller data exchanged for each flow

corresponding `flow-mod` packet (including the Ethernet PDU for fair comparison). As the "match" and "action" fields increase from Type-1 to Type-2, the size of per-flow data increases for both OpenFlow versions. By comparing the two right-most columns, the inter-controller data exchanged for each flow appears comparable with the size of the corresponding flow-mod packet. The difference is due to the different

Table 2.6: Experimental results due to the modification of flow store

OpenFlow Version	Flow Type	Per-flow exchanged data [bytes]	
		OpenFlow packet	Inter-controller data
1.0	Type-1	146	110
	Type-2	218	304
1.3	Type-1	154	110
	Type-2	458	409

internal format[1] and the packet overheads. Type-1 always corresponds to 110 bytes for each flow in the inter-controller traffic due to the same internal representation in ONOS, whereas Type-2 shows a different size depending on the OpenFlow version due to the different match and action fields that are exploited. Thanks to the larger number of available fields, each flow can require up to 409 bytes to be synchronized across the other controllers.

To understand the practical impact of the above experimental results, we observe that the adopted full update scheme in the flow store may generate large synchronization traffic among the controllers, especially when the flow table is large. Thus, we now evaluate the maximum traffic generated in some commercial switches assuming that (i) the flow table is full, (ii) at least one flow modification occurs every 2 seconds, (iii) OpenFlow 1.0 Type-2 flow definitions are adopted in each flow update. Thus, a table update is triggered every 2 seconds, requiring the exchange of the full flow table, and each flow entry corresponds to 304 bytes, based on the results of Table 2.6. We consider the physical OpenFlow 1.0 switches analyzed in [25] with the maximum number of flow rules specified in Table 2.7, where we also show the numerical results for the worst-case inter-controller traffic due to flow updates evaluated based on the previous assumptions. Notably, the bandwidth required to backup the flow table is in the order of Mbps which is relevant, due to just one flow update every 2 seconds.

The above experimental results can be used to compute the inter-controller traffic due to the changes in a flow table for a network, with an arbitrary number of controllers and domains. For this purpose, the master and the first slave controller of each switch must be known, along with the information regarding the existing flows in the switch.

[1]OpenFlow adopts Extensible Match (OXM) representation [24] to allow variable “match” field in the south-bound.

Table 2.7: Maximum inter-controller traffic generated in some commercial OpenFlow switches due to flow store updates

Commercial Switch	Maximum flow rules	Maximum inter-controller traffic [Mbps]
Dell PowerConnect 8132F	750	0.91
HP ProCurve 5406zl	1500	1.83
Pica8 P-3290	2000	2.43

2.6 Distributed Host Store

We describe the impact of the presence of hosts in the network on the inter-controller traffic. Events generated due to hosts in data plane have a transient effect on the inter-controller traffic. This is due to the fact that host information is exchanged among the controllers in a strongly consistent manner backed by RAFT consensus protocol; thus we classify this traffic as *event-driven with incremental updates.*

2.6.1 Methodology

The event according to which a host is added to a switch controlled by ONOS impacts on two data structures. First, an additional port is added to the data structure representing the switch in the topology store. Second, the information about the new host is recorded in the host store. Thus, the inter-controller traffic is affected by two different protocols: the anti-entropy for the topology store, generating periodic and full updates, and RAFT for the host store, generating event-driven and incremental updates. In order to distinguish between the two contributions, we adopt the following methodology.

We exploit the north-bound REST APIs to connect multiple hosts (distinguished by different MAC addresses) to the same port of the switch. In such a way, we avoid adding a new port to the switch for each new host. We actually define a dummy switch at which all the hosts are connected, and in this way we avoid to use Mininet as network emulator. We proceed by simultaneously adding a batch of hosts to the same dummy switch. By evaluating the traffic increment due the transient phase induced by the hosts addition, we evaluate the average amount of data exchanged between the controllers for each new added host. The experiments are carried out for 2 and 3 controllers. Each experiment is repeated 100 times.

2.6.2 Experimental results for 2 controllers

We start by considering the scenario with 2 controllers. Fig. 2.19 shows the result of inter-controller data per host, with the hosts added to controller A. The results depend on the specific role of the controller (leader or follower of the corresponding shard) that acts as master of the switch at which the hosts are added. Notably, this role cannot be set a priori and changes randomly for each experiment. By changing the number of added hosts in a batch the results are the same, thus our numerical results appear to be reliable. By observing the inter-controller data for each host (denoted as D) for different number of hosts that are added in batch, we identify three different behaviors which depend on the roles of the controllers in managing the shards:

- Case 1: Controller A is the leader of all shards of the host store. From the graphs, $D_{\mathrm{A}\to\mathrm{B}} \approx 1000$ bytes and $D_{\mathrm{B}\to\mathrm{A}} \approx 500$ bytes.
- Case 2: Controller B is the leader of all shards of the host store. Both $D_{\mathrm{A}\to\mathrm{B}}$ and $D_{\mathrm{B}\to\mathrm{A}} \approx 2000$ bytes.
- Case 3: Each controller is the leader of at least one shard. $D_{\mathrm{A}\to\mathrm{B}} \approx 1500$ bytes and $D_{\mathrm{B}\to\mathrm{A}} \approx 1200$ bytes.

Recall that controller A is always master of the switch to which the hosts are added. The different data values obtained in the 3 scenarios are explained in the following paragraphs. The results in Fig. 2.19 are grouped based on the above cases.

Case 1 and 2 are the most interesting as they give an upper and lower bound respectively to the per-host data. In case 1, controller A is master and manages directly the host updates received from the switch. Controller A is also the leader for all the shards and thus directly updates the follower controller B, which corresponds to the minimum amount of exchanged data. This is clear from the protocol behavior as shown in Fig. 2.20a: once the leader receives information about the host from REST API, it sends this to the follower. The follower adds this instruction to its log and sends a message to the leader that it is updated. The leader then sends a “commit done” message to the follower to end this transaction.

Instead in case 2, controller B is the leader of all the shards. Thus when a host is added to the switch whose master is controller A, acting as follower for the host store,

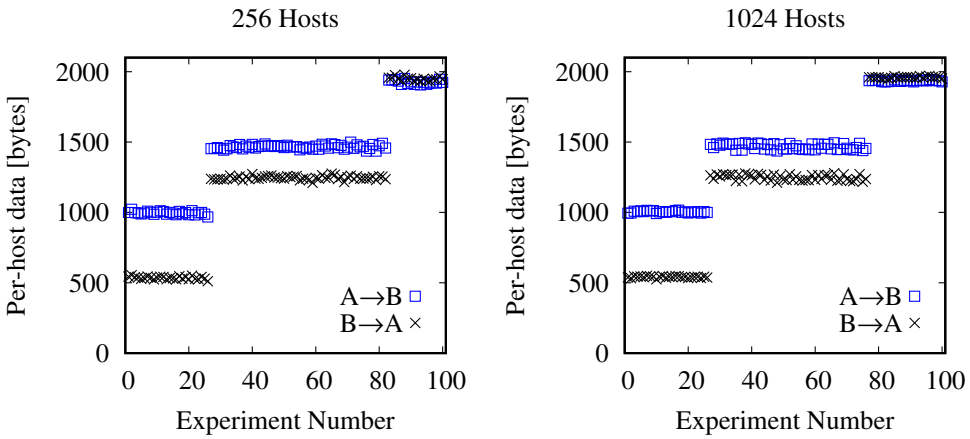
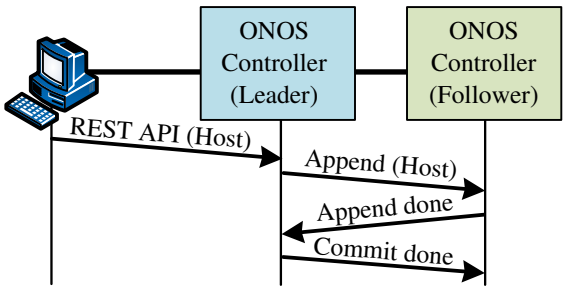
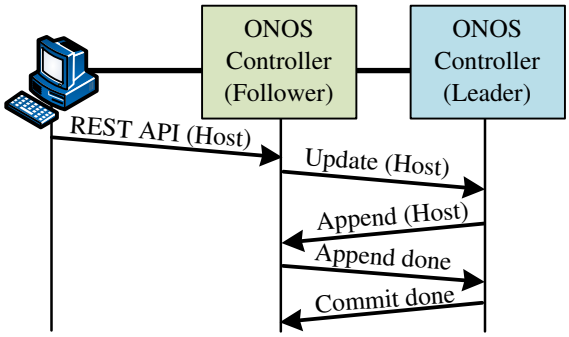


Fig. 2.19: Per-host data exchanged as part of inter-controller traffic between 2 controllers, all hosts added to the switch whose master controller is A



(a) All hosts added to the leader controller



(b) All hosts added to the follower controller

Fig. 2.20: Scenario with 2 controllers for distributed host store

then A must update the leader B first before anything is committed, as shown in the protocol diagram in Fig. 2.20b. After B is notified, the same sequence of messages is observed as in Fig. 2.20a. The additional messages exchanged in case 2 explain the

larger traffic with respect to case 1. Observe now that the actual experimental values are not consistent if only a single message was added in case 2 (denoted "Update host" in Fig. 2.20b). For example in case 1, the follower sends around 500 bytes per host to the leader while in case 2, it sends around 2000 bytes. This can be explained as the Network Configuration Subsystem (i.e. a ONOS internal module) is involved whenever a host is added to a controller. When a controller receives a host to be updated, the Network Configuration Subsystem does a read operation on a strongly consistent data structure backed by RAFT. This read operation is done on the leader. If the controller is itself the leader, the read operation is served locally as in case 1. Instead, the read operation is served remotely by the leader in case 2. This adds extra messages to the inter-controller traffic, which amounts to extra data exchanged per host.

2.6.3 Experimental results for 3 controllers

We now consider the scenario with 3 controllers. Since the results in the previous scenario with 2 controllers depend heavily on the role of the controllers, we adopt the following methodology, in order to just find an upper and lower bound on the inter-controller traffic.

In each experiment, we start all the containers with ONOS controllers and then check if one specific controller is by chance the leader of all partitions of all the data structures, by following the logs of the leader election phase in RAFT consensus protocol; otherwise the containers are rebooted. The 3-controller cluster includes a leader and two follower controllers F_1 and F_2. Two specific cases are adopted to achieve a lower and upper bound on amount of data exchanged per host:

- Case 1: All hosts are added to the switch whose master is the leader controller.
- Case 2: All hosts are added to the switch whose master is one follower controller (assume F_1).

Similarly to the scenario with 2 controllers, case 1 produces the minimum amount of inter-controller traffic due to host addition, whereas case 2 the maximum one.

The data exchanged for each host update is shown in Fig. 2.21. The results show that, regardless of the role of the controllers, the minimum amount of data for each

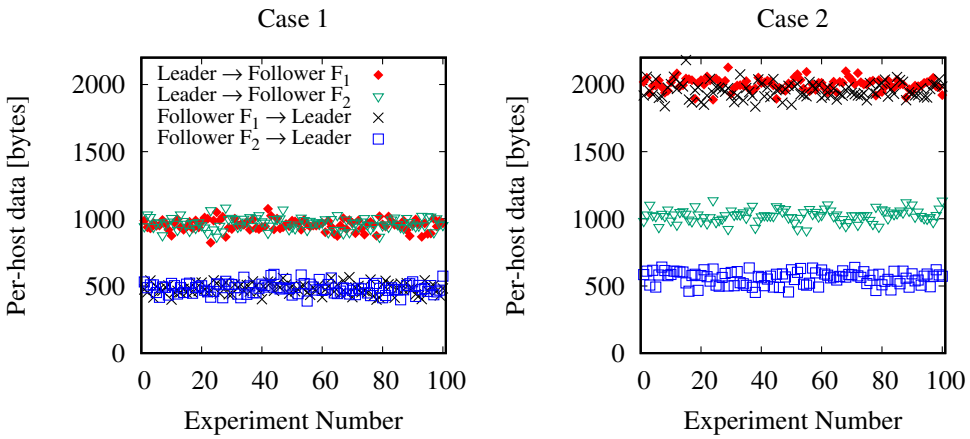


Fig. 2.21: Per-host data exchanged as part of inter-controller traffic among 3 controllers, all hosts added to the leader (case 1) or to one follower (case 2)

flow is around 500 bytes and the maximum one is around 2000 bytes, coherently with the previous scenario. To understand the actual values, we observe the protocol diagrams for the RAFT messages for the two cases, as reported in Figs. 2.22 and 2.23. Case 1 for 3 controllers achieves the same amount of data for each flow than in the 2 controllers case. This is because the data structure is updated in the same manner. Case 2 on the other hand has a different behavior. Fig. 2.21 shows that the data between the leader and F_1 is same as that of case 2 for 2 controllers; on the contrary, the data between leader and F_2 is equivalent to add a host to the leader. This is due to the fact that the Network Configuration Subsystem comes into play in the communication between leader and F_1, since F_1 does a read operation while accessing the leader. This read operation is not done on the communication between the leader and F_2. The traffic between the two followers F_1 and F_2 in all cases does not vary as all the read and write operations in RAFT are done through the leader.

Notably, in the RAFT implementation of ONOS, no more than 3 controllers constitute a partition. Thus the lower and upper bound of data exchanged per host within a partition computed in the previous section are expected to hold in general, independently from the number of controllers in the cluster.

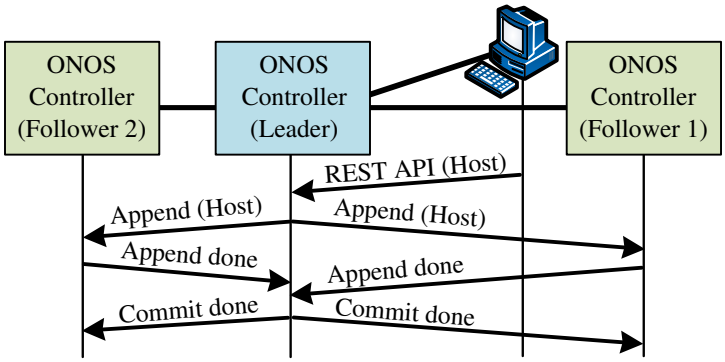


Fig. 2.22: Scenario with 3 controllers and all hosts added to the leader controller

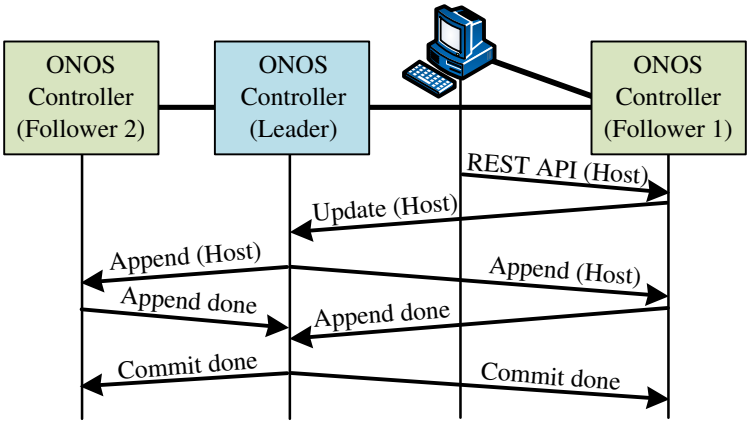


Fig. 2.23: Scenario with 3 controllers and all hosts added to the follower controller F_1

2.7 Related work

The need of a distributed SDN architecture has been thoroughly advocated in literature, since it provides resiliency and scalability as compared to a centralized single controller implementation. Onix [26] is a type of distributed SDN controller which uses partitioning to introduce scalability in distributed SDN. In Onix, although the network topology information known as Network Information Base (NIB) is fully replicated on all the cluster nodes, in a way similar to the topology store in ONOS, yet the information such as forwarding state of devices and link utilization levels are partitioned among controllers, similarly to what RAFT does in ONOS.

Consistency issues in distributed SDN data plane have been highlighted in reference [12], which extended the classic CAP theorem for distributed systems. Reference [12] discusses examples of network policies operating under network

partitions and highlights the advantages of an in-band control plane in distributed SDN controllers. Indeed, out-of-band control information among the controllers may provide less resilience than in-band control one. The intuitive idea is that, for pure in-band control, only in case of data plane partitions the controllers are actually partitioned. Instead, in case of out-of-band control plane, it may happen that the data plane is fully working whereas the control plane is partitioned, creating inconsistency problems. Our work is strongly motivated by the need to understanding deeply the bandwidth required to implement an efficient transportation of in-band control information, and this is crucial for network planning purposes.

Recently, authors in [27] proposed a centralized in-band synchronization approach to achieve a consistent behavior across distributed controllers. Coherently with the motivations of our work, the authors advocate the use of in-band control signaling and highlight the importance of achieving consistency among the controllers. They propose a new set of atomic primitives to ensure consistency, and our proposed experimental methodology could be applied also to their consistency system to evaluate the actual inter-controller traffic due to in-band synchronization. Such evaluation is currently neglected in their work.

The importance of preserving consistency in shared data structures across SDN controllers was highlighted by authors in [28]. In contrast to ONOS as well as Onix, only a global strongly consistent key-value based data store is employed, since it can provide acceptable performance as well as fault-tolerance. The data store is based on replicas which employ state machine replication using a combination of Paxos (a well known consistency algorithm) and Viewstamped Replication (VR). Similar to RAFT, all operations are coordinated using a leader acting as the primary replica which handles all the read and write operations. While comparing the performance, the authors state that existing strongly-consistent data stores implementing the Paxos/VR protocols can perform as good as an eventually consistent data store in Onix for some applications, but the cost of latency is inevitable as a strongly consistent data store is involved.

DISCO (Distributed Multi-domain SDN Controllers) [29] discusses specifically the inter-controller traffic, which is given by two contributions: (i) delegating functions among various agents such as controller reachability, monitoring and relaying inter-controller link health, new controller domain discovery or reservation of inter-domain flow setup and teardown (ii) operating a *Messenger* module for

inter-controller communication based on an Advanced Queuing Messaging Protocol (AQMP) used by the agents. An interesting mechanism in DISCO reconfigures the inter-controller links to abandon congested or slow links and use other controllers as relays for inter-controller communications, which is absent in ONOS.

Notably, references [28] and [29] do not evaluate the cost in terms of bandwidth needed to support the proposed consistency schemes, and our methodology can be adapted to address such issue in both scenarios.

The work in reference [30] investigated the synchronization cost due to the exchange of inter-controller traffic among controllers by analyzing the synchronization delay. This delay consists of the time taken by a controller to detect an event in its domain till the time a different controller becomes aware of it. There exists a trade-off between the synchronization delay and the amount of synchronization data. Different network applications may require faster coordination among controllers at the expense of higher synchronization data exchange rate and vice versa. In contrast, our work focuses on the inter-controller traffic due to network events, while neglecting the delay to achieve consistency. Nevertheless, our empirical models enable a proper planning of the network supporting the control plane, and thus allow to control the corresponding delay performance.

Finally, our approach is perfectly complementary to the work in reference [31], since the latter work focused on the OpenFlow traffic exchanged by ONOS controller with the switches on the south-bound interface. Thus, by combining the results in reference [31] with the results of our work (i.e. the control traffic exchanged among the controllers), it is possible to properly plan and design the whole transport network supporting the overall control plane in a cluster of ONOS controllers.

2.8 Summary

We considered a distributed SDN architecture in which a cluster of ONOS 1.4 controllers, manages all network devices. We focused our investigations on the traffic exchanged between the controllers, which is mainly due to the consensus protocols enabling a consistent view of the network state.

We adopted an experimental testbed based on a cluster of 2 and 3 ONOS controllers and evaluated experimentally the inter-controller traffic due to different

shared data structures and to different network configurations and events. We investigated specifically all the distributed stores that describe the network state (i.e. topology, host and flow stores) and derived some quantitive models to estimate the inter-controller traffic under very general conditions. Even if the results are specific of the considered version of the controller, our methodology is general and can be applied to other versions of ONOS and to different SDN controllers from ONOS. As a future work, we plan to apply our methodology to a larger number of controllers adopting the latest version of ONOS.

Thanks to our experimental results, a network designer can design and plan carefully the network infrastructure that support the inter-controller data plane. This is of paramount importance for network operators running large SDN networks, like SDWANs, where the control data is typically in-band and share the same resources devoted to the customers.

Chapter 3

Time-synchronized operations for software-defined elastic optical networks

Part of the work presented in this chapter has been published in:

- A. Bravalheri, M. G. Alabarce, A. S. Muqaddas, P. Giaccone and A. Bianco, "Experimental validation of time-synchronized operations for software-defined elastic optical networks". In: *IEEE/OSA Journal of Optical Communications and Networking* 10.1 (Jan. 2018), pp. A51–A59.
- M. Garrich, A. S. Muqaddas, P. Giaccone and A. Bianco, "On the importance of time-synchronized operations in Software-Defined electronic and optical networks". In: *2017 19th International Conference on Transparent Optical Networks (ICTON)*. July 2017, pp. 1–4.
- A. S. Muqaddas, M. Garrich, P. Giaccone and A. Bianco, "Exploiting Time-Synchronized Operations in Software- defined Elastic Optical Networks". In: *Optical Fiber Communication Conference.* Mar. 2017, pp. W4J.6.

3.1 Motivation

Traffic volumes in carrier networks keep growing dramatically, driven by the proliferation of high-bandwidth services and applications. To address this challenge, Elastic Optical Networks (EON) enable an efficient use of spectrum resources valuable to extend the lifetime of already deployed optical fibers [32]. EON performs flexible frequency allocation in the network using reconfigurable optical add/drop multiplexers (ROADMs) [33] and bandwidth-variable transmission techniques [34]. In particular, EONs use the spectrum resources of the data plane following the guidelines reported in the ITU-T Recommendation G.694.1 from 2012 [35]. However, the migration from classical wavelength division multiplexed (WDM) fixed-grid spectrum allocation towards flexible EON may require notable long-term investments [36] or gradual migration of the wavelength selective switch (WSS) equipment [37]. Moreover, telecom operators face operational challenges in order to manage such a diverse multi-technology scenario which may also include multi-vendor equipment interoperability issues [38]. In more detail, [38] reports a demonstration of interoperability between multi-vendor optical equipment with the need to adapt several interfaces just to perform an experimental end-to-end resource provisioning. Indeed, these operational challenges may endanger the potential deployment of next-generation flexible-rate transponders and EONs [39].

To address these challenges, novel SDN approaches [40] enable advanced control and configuration features suitable for the breakthrough technologies of the EON data plane. Although legacy GMPLS/Path Computation Element (PCE) architectures already offered a fully separated control plane from the data plane, SDN enhances network programmability via open programmatic interfaces, reduces vendor lock-in issues, and permits innovation and evolution of the network infrastructure [1]. In particular, academic initiatives to control optical components recently proposed open YANG models [41] for EON [42]. In this research direction, YANG models have been proposed for monitoring functionalities in EONs [43] and specific models to manage sliceable transponders [44]. More recently, specific NETCONF protocol features and YANG models have also been proposed to address optical network failure issues [45]. On the industrial side, the recent OpenROADM standardization initiative [46], proposes an interface for multi-vendor ROADM access and configuration based on YANG models. More specifically, OpenROADM targets the

disaggregation of traditionally proprietary ROADM systems and SDN-enablement of traditionally fixed ROADMs.

In the EON data plane, routing and spectrum assignment (RSA) schemes allocate lightpaths ensuring that a set of frequency slots (FS) are *continuous* throughout the routing path [47]. Connections in EON are established (and removed) dynamically, thus potentially leaving sparse FS that become difficult to use by the RSA to reduce blocking probability. This fragmentation problem has been recently addressed with hitless defragmentation techniques able to reallocate the lightpath frequencies without traffic disruption. Examples are the push-pull technique [48], which allows spectrum retuning only over contiguous vacant FS from the source to the destination frequency; and the hop-retuning technique [49], which strictly requires a number of photodetectors equal to the number of FS. Due to system complexity, the former technique is usually preferred over the latter. However, even with RSA schemes combined with push-pull [50], high-load scenarios may drive the network towards the so called "end-of-line situations" limiting the potential benefits of EON [51]. *End-of-line* situations are defined by [51] as cases in which a lightpath obstructs push-pull spectrum defragmentation or non-continuous vacant FS contribute to network blocking. These situations require lightpath rerouting to exploit the remaining capacity not being used by existing RSA and defragmentation techniques, albeit in a non hitless manner.

In this context, SDN could be exploited to address this challenge. In particular, time-synchronized operations (TSO), have been recently proposed in the form of southbound protocol extensions to coordinate distributed operations simultaneously [52]. Indeed, TSO are gaining interest in the research community as an SDN feature capable to improve network performance [53] and to develop novel applications [54].

3.1.1 Our contributions

In this chapter, we present our proposal [55] of TSO for EON to address end-of-line situations efficiently performing lightpath rerouting to minimize the disruption time. We discuss the implementation of our proposal using the existing protocols, and we show the benefits in a test scenario comparing performance against traditional asynchronous operations [56]. We present the experimental validation of TSO for

lightpath reassignment in a five-node metropolitan optical network test-bed. We compare the network performance in terms of optical signal-to-noise ratio (OSNR) and optical power budget between our TSO-based approach and the traditional asynchronous technique. We observe that both techniques maintain a similar network performance, thus validating the convenience of the TSO-based approach given its reduction of disruption time. After presenting the experimental validation, we review some recent literature on TSO applicability in both electronic and packet optical networks.

3.1.2 Organization of the chapter

The remainder of this chapter is organized as follows. Sec. 3.2 presents an end-of-line scenario in optical networks, which involves disruption of lightpaths to reroute them. Sec. 3.3 elaborates on time-synchronized operations for EON. Sections 3.4 and 3.5 present the analytical evaluation and experimental validation of our TSO-based solution respectively. Sec. 3.6 discusses relevant related work and Sec. 3.7 summarizes the chapter.

3.2 An end-of-line scenario: non-continuous vacant FS

In this section, we provide an example of end-of-line situation to illustrate the need for lightpath rerouting to better exploit the remaining optical spectrum resources. Then, we detail the traditional asynchronous technique commonly employed in non SDN-enabled networks to address these situations

Fig. 3.1 shows an example of end-of-line situation due to non-continuous vacant FS in a network. The sample topology in the example consists of 5 nodes: A, B, C, D and E, which are connected by 5 links, assuming 6 FS per link (number of FS chosen to simplify the explanation). Initially, assume that there are 4 lightpaths in the network:

- L_1: from A to D via C, requiring 2 FS
- L_2: from A to D via C, requiring 3 FS

- L_3: from A to D via B, requiring 3 FS
- L_4: from E to D via B, requiring 2 FS

Thus 1 FS is available in both *A-B-D* and *A-C-D* paths. Let us assume a new lightpath requests 2 FS from A to D. Note that defragmentation would not increase the available FS in each link to accommodate this new lightpath. Therefore, either this new request is rejected or existing lightpaths need to be rerouted. The latter case is preferred, as shown in Fig. 3.2, because it reduces the network blocking probability. Rerouting in Fig. 3.1 requires swapping lightpaths to achieve the configuration in Fig. 3.2.

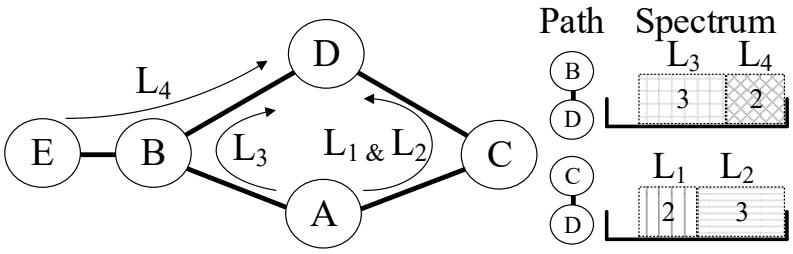


Fig. 3.1: Topology with lightpaths in an end-of-line scenario

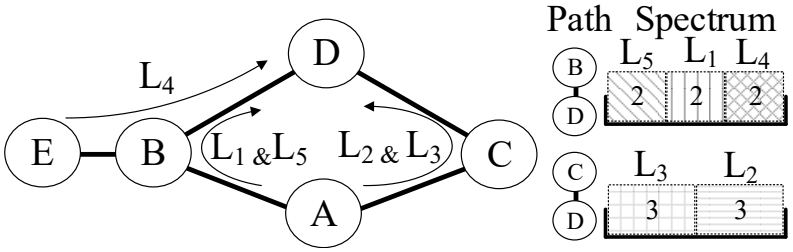


Fig. 3.2: Re-routing to accommodate a new lightpath

We define as *asynchronous* (ASY) approach the technique that executes the operations asynchronously as depicted in Fig. 3.3a. The ASY approach addresses the end-of-line situation shown in Fig. 3.1 to achieve the network state in Fig. 3.2 performing the following four operations. First, L_3 is disrupted sending *tear-down* requests to all the nodes. Second, L_1 is rerouted from *A-C-D* to *A-B-D* with two commands *tear-down* and *setup* for its migration. Third, L_3 is now *setup* in its new route *A-C-D*. Finally, the new lightpath L_5 can be allocated on *A-B-D* and the network state depicted in Fig. 3.2 is achieved. Note that this operation of lightpath swapping implies a non-negligible disruption time for both L_1 and L_3.

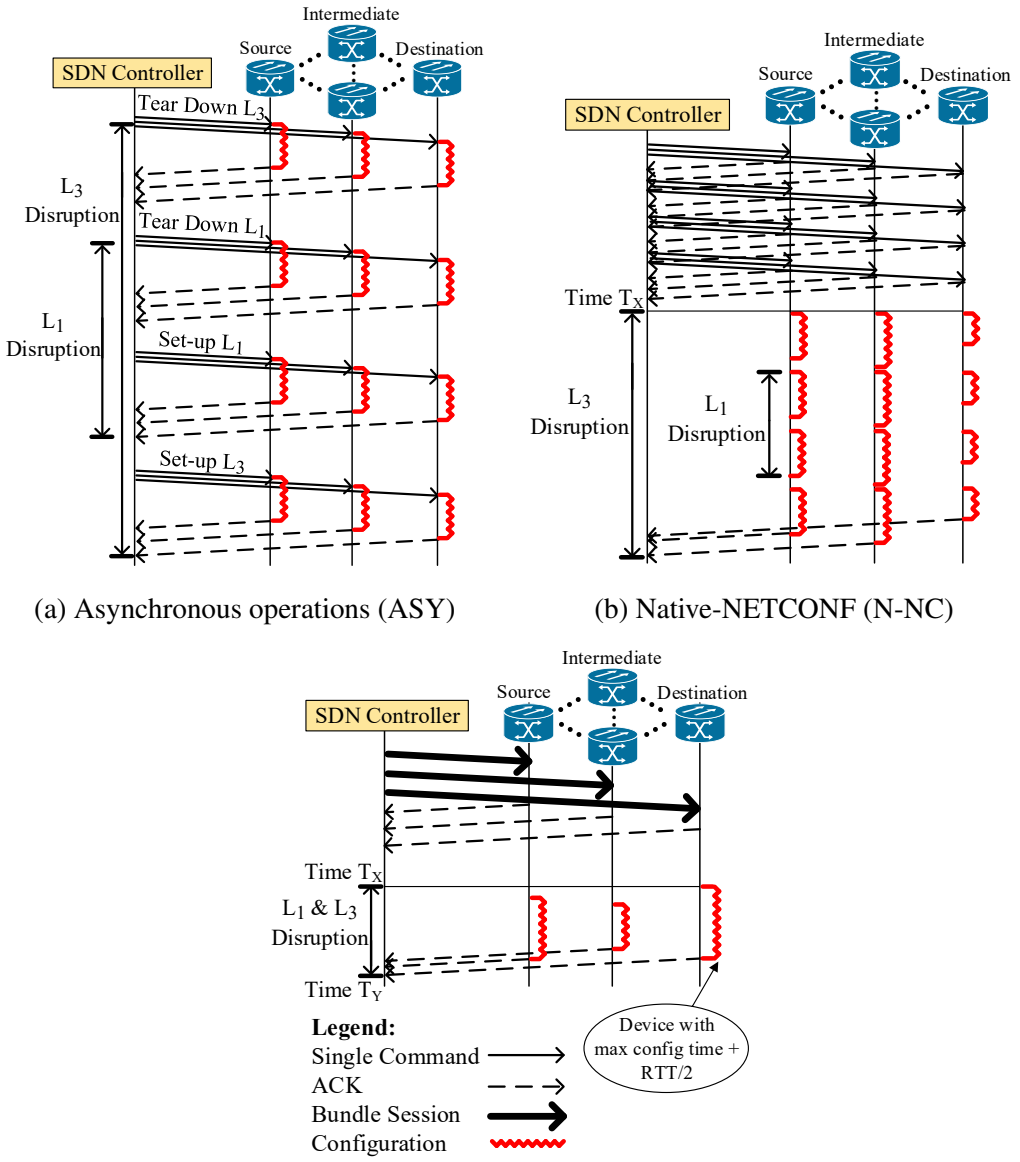
Nonetheless, it worthwhile mentioning that differently from the sequence illustrated in Fig. 3.3a, the reassignment of L_1 could be performed without any disruption time just by implementing Make-before-Break (MbB) technique as specified in RSVP-TE [57]. In particular, given that spectrum resources are made available using overprovisioning in the *A-C-D* path by tearing down L_3, L_1 can be setup in this new route before tearing down its initial allocation in *A-B-D*. However, note that MbB for L_1 does not reduce the disruption time for L_3.

3.3 Time-synchronized operations for EON

In this section, we present our proposal [55] of TSO for EON which leverage on recently provided features in SDN. Simultaneous operations can be coordinated using timestamps within industry-standard southbound configuration messages. In the case of lightpath swapping, our approach operates as shown in Figs. 3.3b and 3.3c.

In case of NETCONF, time extensions to the protocol have been recently published as an RFC [58]. The SDN controller sends a *scheduled-RPC* message to the optical node to execute an operation at a specific time. Note that NETCONF does not provide the capability to bundle operations natively. Therefore, one command per operation is issued and scheduled using timestamps in a sequential manner accounting for the configuration time as shown in Fig. 3.3b (i.e. four operations same as ASY). We refer to this implementation as Native-NETCONF (N-NC). Indeed, similarly as for the ASY case, the commands required to reroute L_1 could be inverted (i.e., setup before tear down) implementing the MbB approach so that L_1 does experiences a negligible disruption time. Nonetheless, an Intelligent Agent can be implemented either at the SDN controller or at the optical node that processes NETCONF (IA-NC) messages to group several operations into a single configuration [59].

In case of OpenFlow (OF), two features are included in its latest version 1.5 [60]: a *bundle* of operations can be executed simultaneously [40], and this bundle can be *scheduled* for execution at a given time, as shown in Fig. 3.3c. The scheduling of the bundle depends on the node with the maximum sum of the configuration time plus the half round trip time (RTT). In Fig. 3.3c, we assume that this is the case of the destination node and it starts to execute the bundle of operations at time T_X and it acknowledges the SDN controller at time T_Y after it finishes its configuration. Given that all other nodes have a configuration time smaller than the destination node, their



(a) Asynchronous operations (ASY)

(b) Native-NETCONF (N-NC)

(c) OpenFlow (OF)

Fig. 3.3: Asynchronous vs. TSO-based approaches in NETCONF and OpenFlow. Source node corresponds to A, intermediate nodes correspond to B and C, and destination node corresponds to D, respectively, in Figs. 3.1 and 3.2.

configuration can be done within the time interval between T_X and T_Y. By doing so, the smaller configuration times in other nodes compared to the maximum case (destination node in Fig. 3.3c) become transparent to the disruption time. Indeed, this relaxes the requirement of full time-synchronization for the OF approach. Bundling

commands in OF requires opening a session by the SDN controller to the optical node with a *bundle-open* message. Thereafter, multiple commands are sent to the optical node to be added to the bundle. This is followed by a *bundle-commit* message to specify the time at which the bundle should be executed. Note that bundling network operations by means of the OF bundling feature differs from launching an application (e.g., script file or program) at the SDN controller that issues multiple commands to a given network node. For instance, multiple WSS configurations for different spectrum filtering patterns could be merged into a single WSS filtering pattern modification within a bundle. However, the approach using the multiple-command application would update the WSS filtering pattern upon receiving each command separately. It is worthwhile mentioning that OF and IA-NC cause the same disruption as both implementations permit to bundle several operations as a single configuration. Hence, we refer to them as OF/IA-NC while evaluating their performance.

The temporal accuracy of the time-synchronized approach depends on the maximum value of two contributions. On the one hand, we consider the *worst-case configuration time* of all optical nodes involved in the reconfiguration. This time depends on several factors including the common coexistence of data-plane devices from different vendors in carrier-grade optical networks, the dependence of the configuration time on current load of the agent at the optical node, aging issues or other random behaviors. However, this worst-case configuration time can be estimated with some error considering the average reconfiguration time with respect to the load [59, 61] (see Fig. 3.4(b)); given a particular vendor, different firmware/software implementation and control plane protocols. On the other hand, *worst-case synchronization error* among devices needs to be taken into account. To this end, local clocks at the optical nodes as well as the SDN controller can be synchronized with a common reference clock using Precision Time Protocol (PTP), or an improved version named ReversePTP [62]. Indeed, the accuracy of up to 1 µs provided by ReversePTP makes its contribution to the TSO inaccuracy negligible compared to optical configuration times which are in the order of seconds [59, 61]. Furthermore, GPS can also be used as backup for the reference clock to enable TSO.

In summary, the efficiency of the TSO approach improves with better time accuracy and better knowledge of the reconfiguration time. In this chapter, given that all the schemes under analysis (ASY, N-NC, OF) are affected by these worst-case

considerations, the current conclusions hold. Consequently, we leave further analyses on these two problems outside the scope of our work.

3.4 Analytical evaluation of TSO

In this section, we evaluate the disruption time of ASY, N-NC and OF/IA-NC, considering the lightpath swapping scenario of Sec. 3.2.

We assume that each node i has a constant configuration time c_i regardless of the operation. The ASY approach is composed of four operations:

- Tear down L_3
- Tear down L_1
- Setup L_1
- Setup L_3

Each operation lasts for t_{op} time as shown in (3.1).

$$t_{op} = \max_i(RTT_i + c_i) \tag{3.1}$$

where RTT_i is the Round Trip Time between the SDN controller and node i. Thus, the total disruption time experienced by lightpath L_3 is shown in (3.2).

$$t_{ASY} = 4 \times t_{op} - \min_i(RTT_i/2) \tag{3.2}$$

where the second term is subtracted because the disruption starts when the nearest node receives the tear-down message from the controller. The N-NC approach concatenates four operations, similar to ASY. Hence, the disruption time for N-NC case is shown in (3.3).

$$t_{N-NC} = 3\max_i(c_i) + \max_i(c_i + RTT_i/2) \tag{3.3}$$

where the second term is due to the last operation in which the controller receives an ACK. The OF/IA-NC permits simultaneous operations, thus the disruption lasts for $t_{OF/IA-NC}$ as shown in (3.4).

$$t_{OF/IA-NC} = \max_i (c_i + RTT_i/2) \tag{3.4}$$

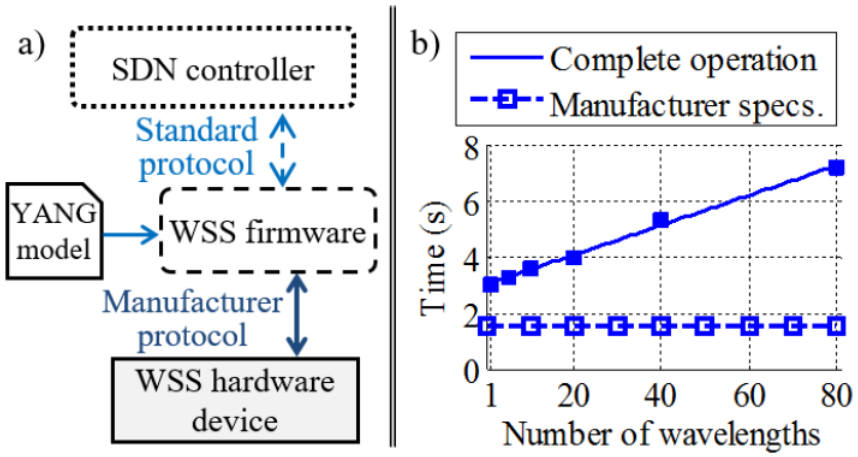


Fig. 3.4: (a) System modules for WSS control in the optical network test-bed. (b) Time required for a single WSS operation vs. the number of wavelengths. Empty squares report manufacturer specifications (upper bound for the WSS hardware configuration time) and solid squares report experimental measurements of a complete operation cycle, i.e., service time (average of ten experiments in two WSS devices)[59, 61].

In order to evaluate the impact of the configuration time of the WSS devices on the lightpath disruption time, we review the reported experimental results [59, 61]. In particular, an SDN controller makes use of a standard protocol (e.g., a REST interface) to communicate with the firmware of the WSS (Fig. 3.4(a)). Leveraging on YANG models, the SDN controller can issue specific requests to the firmware of the WSS. For example, the attenuation of any given WSS device at any desirable position of the optical spectrum can be arbitrarily set by the controller. Fig. 3.4(b) reports the time that is required to perform a change of attenuation in the WSS device as a function of the number of wavelengths for which the attenuation is being adjusted. More specifically, the empty squares report the time requirement as specified by the WSS device manufacturer. The solid squares report the time required to complete the operation inclusive of the control signaling, the firmware execution time and the WSS operation. (The signaling propagation time between the SDN controller

and the optical node is negligible.) During the experiment, the applied attenuation for a given group of wavelengths is changed from maximum to minimum and vice versa. The number of wavelengths being switched is varied. Results reported in Fig. 3.4(b) are the average of ten experiments in two WSS devices and exhibit a linear dependence on the number of channels to be configured with minimal standard deviation (confidence intervals using vertical lines are not reported for the sake of legibility) as in [61]. The curve indicates that the time required to complete the adjustment of the WSS-applied attenuation is proportional to the number of wavelengths for which the attenuation is being adjusted.

Considering the above reported experimental results that characterize the WSS operation time, Fig. 3.5 shows the disruption time as a function of $\max_i(c_i) \in [3,7]$ and number of wavelengths $\in [1,80]$ with constant RTT. Note that the proportional dependence between the configuration time required by a WSS and the number of wavelengths it is required to adjust permits the double x-axis depicted in Fig. 3.5. OF and IA-NC outperform ASY and N-NC as they bundle all the operations in a single configuration instead of four, thus reducing the disruption time by 75% in this case. The difference between ASY and N-NC is minimal and cannot be observed in Fig. 3.5. This is due to the fact that N-NC, as shown in (3.3), includes the term $RTT/2$; whereas ASY, as per (3.1) and (3.2), includes the round trip time as $3.5 \times RTT$. Thus the difference of RTT between ASY and N-NC is $3 \times RTT$, which amounts to 30ms in Fig. 3.5 and is thus negligible as compared to the overall disruption time.

Finally, Fig. 3.6 explores the disruption time for a constant $c_i = 50$ms, $\forall i$, small enough to observe the impact of the $\max_i(RTT_i)$. Note that a $c_i = 50$ms is consistent with Micro-Electro-Mechanical systems (MEMS) technology employed in fiber switches [63][64]. As in the previous analysis, the bundling feature in OF and IA-NC reduces the communication rounds between the optical nodes and the controller, thus reducing the disruption time due to RTT. Consequently, as RTT increases, the reduction grows from 75% to 83.3% when comparing OF and IA-NC against ASY. Furthermore in this case, N-NC performs better than ASY but worse than OF/IA-NC. Figs. 3.5 and 3.6 show that the disruption time reduction of 75% is minimum.

Different configuration time for ASY/N-NC vs OF/IA-NC Note that Fig. 3.4(b) implies that the configuration time c_i for OF/IA-NC (in (3.4)) may be more than ASY

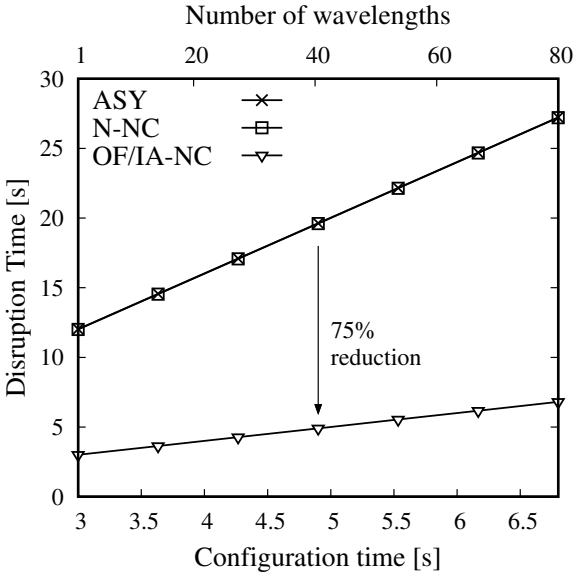


Fig. 3.5: Disruption time for variable number of wavelengths with $RTT = 10$ ms

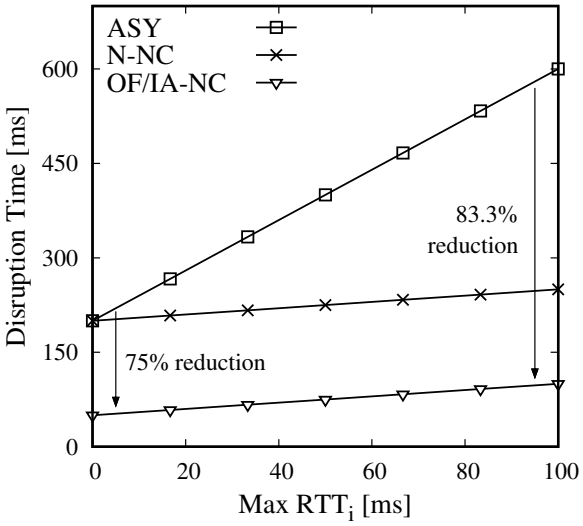


Fig. 3.6: Disruption time for two different scenarios

and N-NC (in (3.2)-(3.3)); since more number of wavelengths are simultaneously configured in the WSS in the OF/IA-NC case. However, still in this case, OF/IA-NC outperforms ASY and N-NC due to bundling of operations; consequently involving more number of wavelengths per WSS operation. For e.g., if a WSS operation requires 1 wavelength, it takes 3 seconds according to Fig. 3.4(b), however with just an increment of 25% in time i.e., 4 seconds, 20 wavelengths can also be configured in

a single WSS operation. To further elaborate on this situation, consider the example of the scenario shown in Fig. 3.1, however this time, we scale all the FS by 10. This means that the capacity of each link is 60 FS. In this case, the lightpaths will have following requirements:

- L_1: from A to D via C, requiring 20 FS
- L_2: from A to D via C, requiring 30 FS
- L_3: from A to D via B, requiring 30 FS
- L_4: from E to D via B, requiring 20 FS

Assuming all the nodes use the same WSS with configuration time according to Fig. 3.4. In this way, we can neglect the RTT for comparison. This implies the following configuration time per operation for ASY and N-NC:

- Tear down L_3: requires 30 channels $\approx$ 4.61 s
- Tear down L_1: requires 20 channels $\approx$ 4.12 s
- Setup L_1: requires 20 channels $\approx$ 4.12 s
- Setup L_3: requires 30 channels $\approx$ 4.61 s
- **Total** $\approx$ 17.46 s

Whereas in OF/IA-NC case, 30 channels need to be configured in all paths, hence it requires $\approx$ 4.61 s, which is a reduction of 73.60% as compared to ASY/N-NC case. This is due to the bundling of commands as discussed earlier.

3.5 Experimental validation of TSO

In this section, we first provide an overview of the five-node metropolitan optical network test-bed where the experiments are performed. Then, we detail the experimental setup that emulates the end-of-line scenario shown in Fig. 3.1. Finally, we report and discuss the experimental results.

3.5.1 Optical network test-bed overview

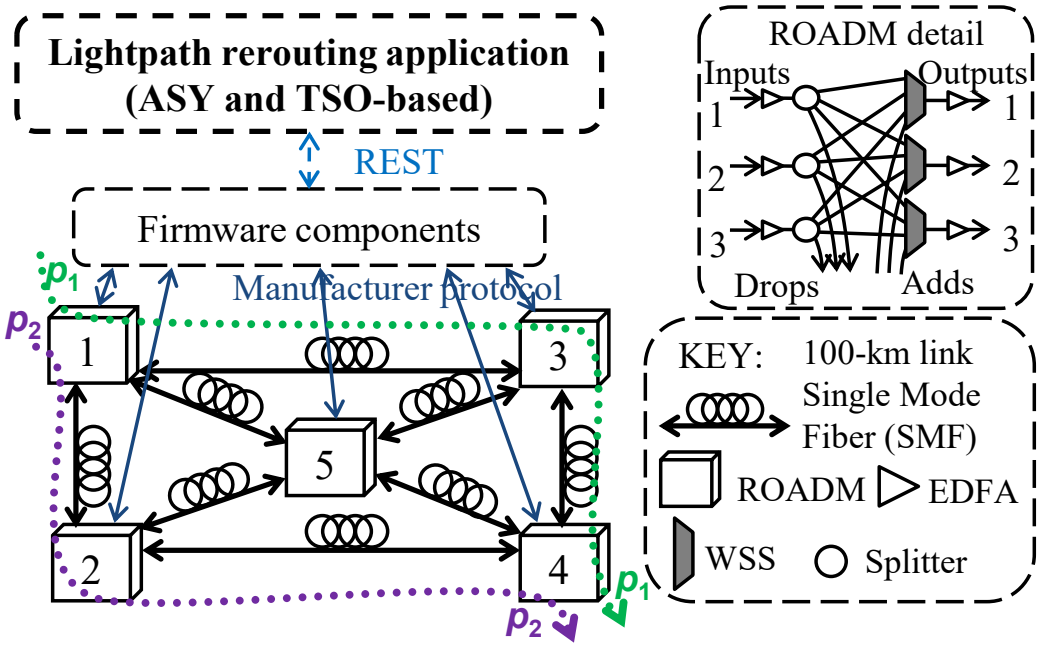


Fig. 3.7: General architecture for the SDN-enabled test-bed.

The experimental results of this chapter are obtained using an SDN-enabled five-node metropolitan optical network test-bed located at CPqD [59]. More specifically, the network test-bed comprises 4 ROADMs of degree 3 and a central ROADM of degree 4 interconnected to form a partial mesh topology using 100-km single mode fiber (SMF) links as shown in Fig. 3.7. The ROADM node architecture is broadcast-and-select (B&S) using one splitter per input port and one WSS per output port. In particular, the WSS devices are from Finisar and belong to its ROADMs & Wavelength Management product portfolio. More specifically, 1×5 Flexgrid® WSSs[1] acquired in 2010 are used in the central ROADM, whereas 1×4 Flexgrid® WSSs[2] acquired in 2012 are used in the ROADMs at the edges of the network. EDFAs are placed at each input and output port to compensate for span and node losses. No physical dispersion compensation modules are used. The transmitter is composed of 80 continuous wave (CW) lasers with 50 GHz channel spacing. Each

[1]Product Code: 10WSPA05ZZL. Discontinued product. Preliminary version of the current 1×9 and 1×20 WSS devices detailed in https://www.finisar.com/roadms-wavelength-management/10wsaaxxfll3

[2]Product Code: EWP-AA-104-96F-ZZ-L https://www.finisar.com/roadms-wavelength-management/ewp-aa-010x-96f-zz-l

CW is modulated by four multiplexed lines of 32 Gb/s (PRBS $2^{31}-1$), obtaining 80 128-Gb/s DP- QPSK orthogonal channels. Transmission impairments and non-linear effects are assumed to be compensated at the receiver (out of the scope of this thesis).

3.5.2 Experimental setup

In order to investigate the approaches described in Sections 3.2 and 3.3 for introducing new connections in an end-of-line situation, two experimental tests are carried out using the metropolitan optical network test-bed. Both tests consist of configuring the network to present an initial state and after it requiring the establishment of a new lightpath.

In the first experiment, the SDN controller is configured to sequentially send commands corresponding to *setup* and *tear-down* operations for each individual lightpath, according to the ASY approach. By contrast, in the second experiment, the SDN controller is configured to send only one command for each piece of equipment, reconfiguring all the lightpaths simultaneously, according to the TSO approach. The messages sent by the SDN controller is done using RESTCONF [65]. It is based on REST API to configure the parameters of a device defined in the YANG model, using the same datastore concepts defined for NETCONF.

During the tests, a set of lightpaths L_n similar to those shown in Fig. 3.1 and Fig. 3.2 are defined, although, the branched topology is replaced by a plain one, with all lightpaths starting at node 1 for simplicity. The set of lightpaths are routed in the test-bed through two link-disjoint physical paths composed by the outermost nodes of the test-bed. In particular, as shown in Fig. 3.7, p_1 traverses nodes 1, 3 and 4; and p_2 traverses nodes 1, 2 and 4. Table 3.1 shows detailed information of each lightpath.

Note that when the DWDM 80-channel comb is launched into the network, the WSS at the first ROADM is used to filter undesired (interleaved) channels in order to generate a 40-channel scenario. This spacing is used to observe the noise power, thus the OSNR can be precisely estimated. Fig. 3.8 shows the sample optical spectrum, where inter-channel spacing is 100 GHz due to filtering interleaved channels. The number of channels of each lightpath can be calculated by (3.5) using the central frequencies of first and last channel from Table. 3.1 and the channel spacing of 100 GHz as shown in Fig. 3.8.

Table 3.1: Lightpath characteristics before (top) and after (bottom) the introduction of L_5. The listed channels are represented by the central frequency.

	No. Channels	Physical Path	First Channel (f_{first})	Last Channel (f_{last})
L_1	13	p_2	192.8 THz	194.0 THz
L_2	20	p_2	194.1 THz	196.0 THz
L_3	20	p_1	192.8 THz	194.7 THz
L_4	13	p_1	194.8 THz	196.0 THz
L_1	13	p_1	193.5 THz	194.7 THz
L_2	20	p_2	194.1 THz	196.0 THz
L_3	20	p_2	192.1 THz	194.0 THz
L_4	13	p_1	194.8 THz	196.0 THz
L_5	13	p_1	192.2 THz	193.4 THz

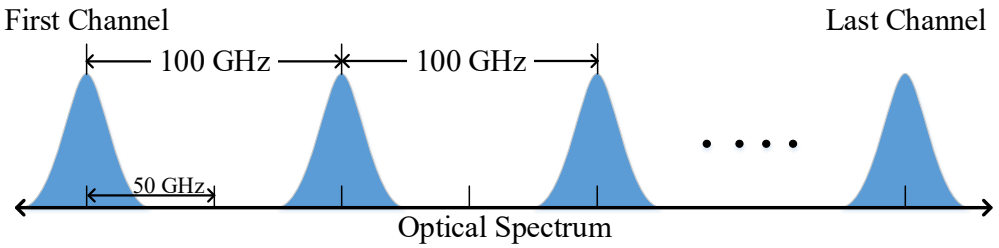


Fig. 3.8: Sample of the optical spectrum in the experiment to illustrate number of channels per lightpath

$$\text{Number of channels} = \frac{f_{last} - f_{first}}{100\ \text{GHz}} + 1 \tag{3.5}$$

In the first experiment, the sequence of the SDN controller actions start at six different moments (likewise Fig. 3.3a):

t_0 – establish initial state

t_1 – tear-down L_3

t_2 – tear-down L_1

t_3 – setup L_1

t_4 – setup L_3

t_5 – setup L_5

On the other hand, in the second experiment, the sequence of the SDN controller actions start at two different moments (likewise Fig. 3.3b):

t_0 – establish initial state

t_1 – reroute lightpaths

After each action of the SDN controller, the optical spectra and powers for all nodes of the network are measured with an optical spectrum analyzer (OSA) at 6 monitoring points as shown in Fig. 3.9. The average OSNR and spectrum tilt (maximum difference of power among all channels) are also calculated at the last node of the physical paths. Since the first node of the path is used to select the input channels, the acquisition is performed after the WSS of this node, and due to the node architecture, the measured power corresponds to $3/40$ of the WSS output power. For the other nodes, the acquisition is performed before the WSS, and due to the node architecture the measured power corresponds to $1/6$ of the amplifier output power. Therefore, different power levels between the first ROADM and the subsequent ones are expected due to the different monitoring points inside the B&S architecture. Finally, it is worth mentioning that the attenuation performed at the WSSs is only applied to route the channel signals across the network, and is not applied to equalize each individual signal power. This choice is meant to better explore the physical layer implications in terms of power tilt across the C-band, and enables us to properly focus on the performance of the TSO-based approach against the traditional asynchronous technique. Future works may combine the current proposal in simultaneous operation with equalization techniques.

Note that here, we report the physical layer in terms of the optical signal performance instead of the disruption time experienced by the lightpaths while performing the lightpath swapping. This is influenced by the fact that in the experiment, the quality of all optical signals is measured after every operation to ensure that it is performed successfully; where 6 operations are involved in the ASY case and 2 operations are involved in the TSO case. As mentioned previously, to check the optical signal performance, the OSA is used to observe the spectrum at 6 different monitoring points at each ROADM (with more details in reference [66], which uses the same testbed). This also includes an optical cross-connect (i.e., optical

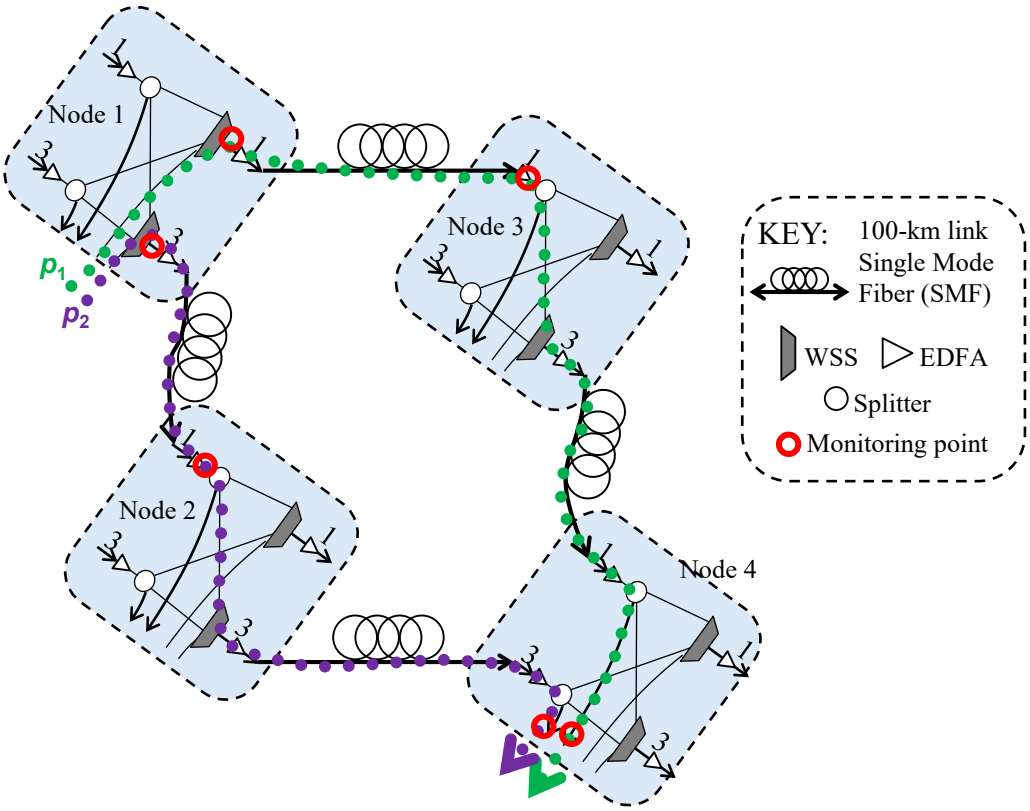


Fig. 3.9: Optical signal monitoring points in the topology

fiber switch) which is used at the input of the OSA to get the spectrum from all monitoring points. Different monitoring points are required so that during and after each operation, the quality of optical signal is checked to verify the success of the operation; however this takes a considerate amount of time since the OSA needs to sweep all the frequencies under observation. For instance, assume an OSA measure takes t_{OSA} seconds. Consequently, given that there are 6 monitoring points in the testbed, after any operation, at least $6 \times t_{OSA}$ seconds are required. Hence we focused on the quality of the lightpath at the receiver in terms of the power, OSNR and the spectrum tilt. Future works may thoroughly investigate the reduction in the lightpath disruption time in an experimental fashion.

3.5.3 Experimental results and discussion

Figs. 3.10 and 3.11 show the optical spectrum of the signal received at the last node of each optical path, for both approaches in the initial and final states. In all charts,

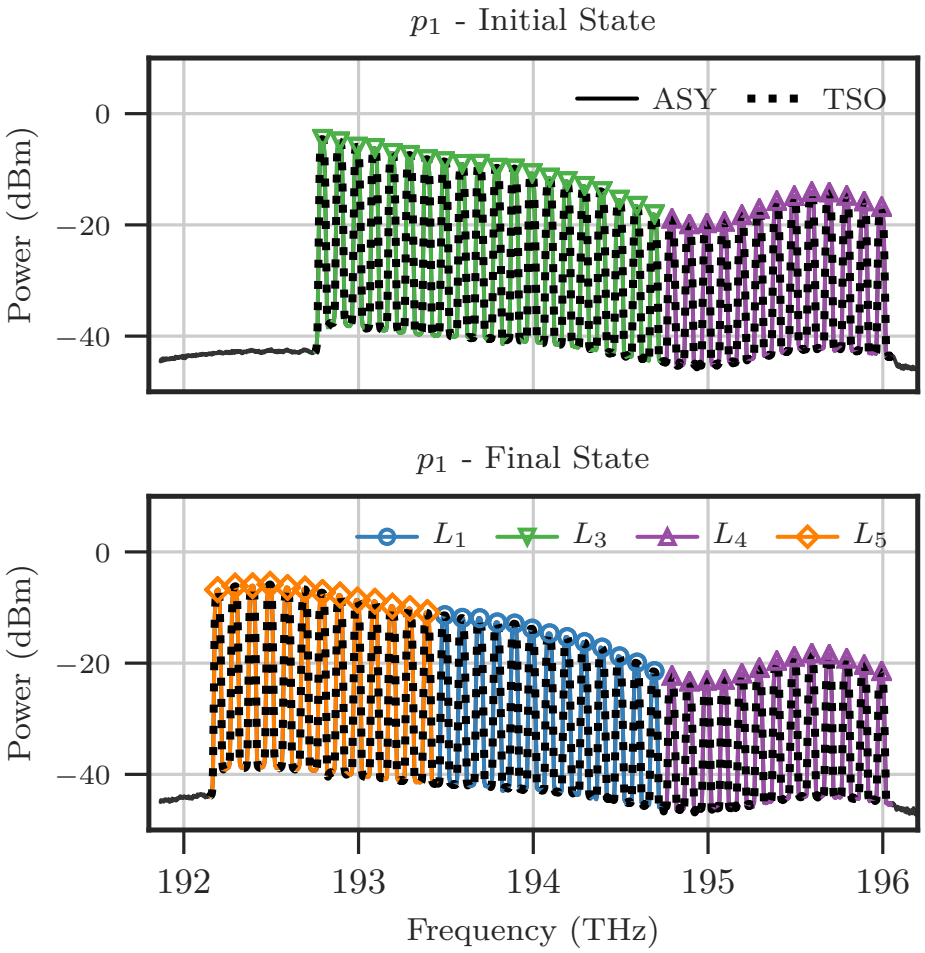


Fig. 3.10: Optical spectrum of the received signal at the last node of p_1, before (top) and after (bottom) the introduction of L_5 for both techniques.

the curve for the TSO-based approach virtually overlaps the curve for the ASY technique. This result was already expected, since the channel configuration is the same, regardless of the technique, before and after the rerouting procedure. Interestingly, the optical power for the individual channels changes after the techniques are applied, as noticeable in the spectral region around 195 THz. These changes are a consequence of the non-linear dynamic behavior of the optical amplifiers whose gain profile depends on the input spectrum shape as a whole, but not only on the input power. A power tilt variation between initial and final states can be observed (but not between the two approaches), since no flattening technique is used neither in the

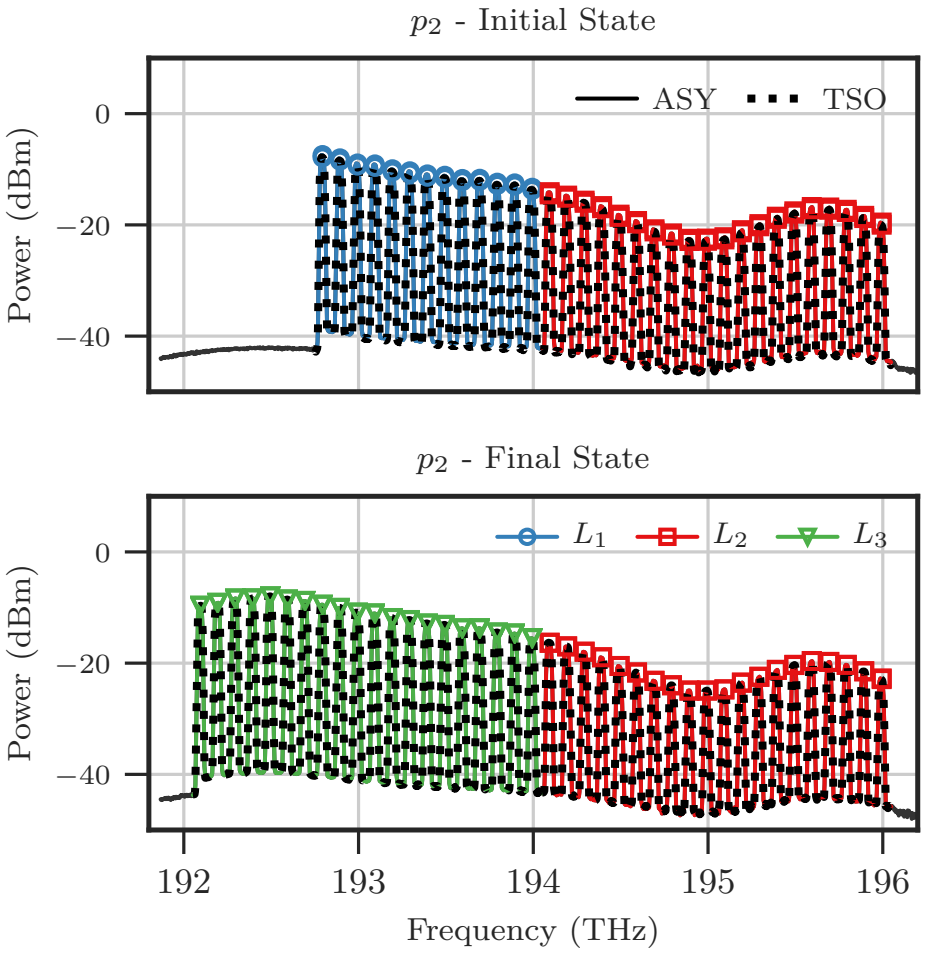


Fig. 3.11: Optical spectrum of the received signal at the last node of p_2, before (top) and after (bottom) the introduction of L_5 for both techniques.

amplifiers nor in the WSSs, and this power tilt also changes after the introduction of the lightpath L_5.

The impression that the chosen technique do not impact in the system performance in terms of signal quality, as suggested by the previous figures, is confirmed by Table 3.2, where performance indicators for the final state of the network are compared. The indicators for both techniques are hardly distinguishable.

Fig. 3.12 illustrates the changes in the optical power for all nodes, after each action of the SDN controller (here represented by the aforementioned time instants t_n). The optical power measured in the first node is 10 dB lower than the other nodes because it is acquired in a different monitoring point, with a different split ratio as

Table 3.2: Comparison of the two methodologies showcasing total power, average channel OSNR and spectrum tilt measured at the final node of the physical paths after the rerouting procedure.

	ASY		TSO	
	p_1	p_2	p_1	p_2
Power (dBm)	4.66	2.81	4.62	2.80
OSNR (dB)	28.11	27.00	28.11	26.98
Tilt (dB)	17.44	16.95	17.47	16.96

previously mentioned. As the total number of optical channels increases with the introduction of L_5 and not all the optical amplifiers are operating under saturation condition, an overall power increasing is experienced between the initial and final states of the experiments. Moreover, during the first experiment, the optical power initially decreases in the first node, due the two consecutive *tear-down* operations, but raises again with the *setup* operations. The curves for subsequent nodes follow this shape, with the exception of node 4 for p_1, clearly due to a saturated amplifier.

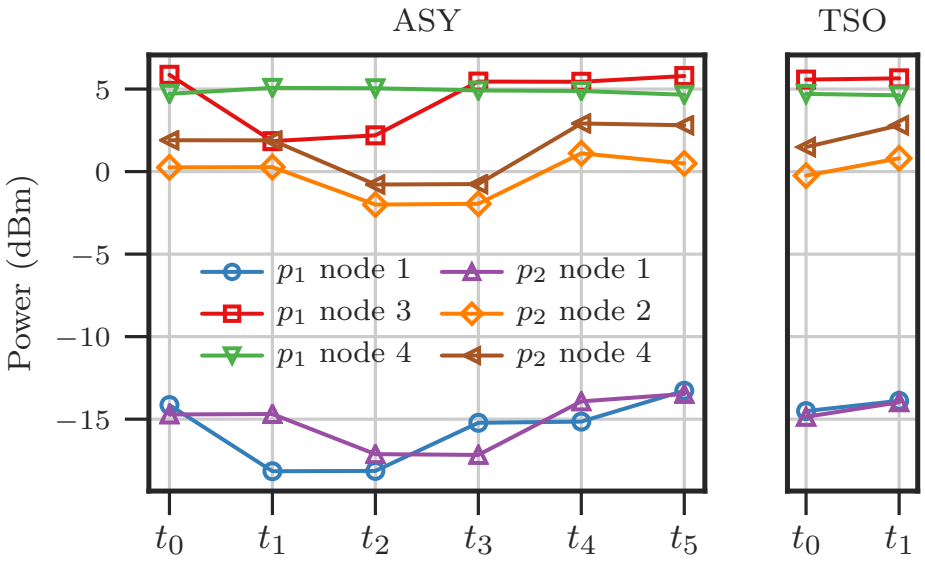


Fig. 3.12: Power fluctuations in each node of the test-bed during the rearrangements for both techniques.

Finally, Fig. 3.13 illustrates the changes in the OSNR of the received signal in the last node after the actions of the SDN controller. In a contrary way to the power

behavior, the overall OSNR trend decreases despite of the intermediary increase in the first experiment. This is also a result of the non-linear dynamic behavior of the amplifier, because with low total input power (i.e., low number of channels) its performance in terms of OSNR improves.

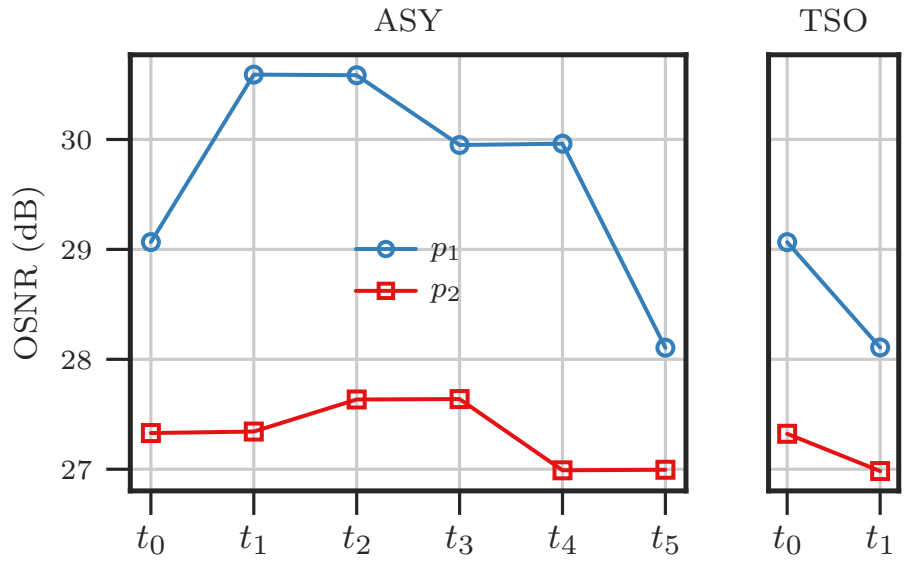


Fig. 3.13: Average OSNR variation during the rearrangements for both techniques.

3.6 Related work

The utilization of TSO is gaining interest in the research community on SDN. One of its applications is our work, as described in previous sections. This section reviews other works which involve TSO in electronic packet and optical networks. In Section 3.6.1, we survey several initiatives in electronic packet networks that employ TSO to improve network performance and enhance monitoring functionalities, thanks to timed network updates. In Section 3.6.2, we review TSO in optical networks that enable a novel security application and our work to reduce lightpath disruption time. We discuss TSO implementation requirements including clock availability in network elements in Section 3.6.3.

3.6.1 Time-synchronized operations in Electronic Packet Networks

In this section, we review three initiatives that employ TSO in electronic packet networks (EPN).

TSO in EPN for flow swapping

SDN provides a global-network view to enable advanced traffic engineering policies. This may require two apparently contradictory objectives: frequent path modifications while avoiding misbehaviors (e.g., packet losses, outages, routing loops). A conventional way to meet these objectives is to ensure spare network capacity. However, this may not be possible in case of high load conditions.

In this context, Mizrahi and Moses [52] propose and implement a TSO approach, referred to as TIME4, to efficiently manage the existing network capacity. Specifically, they target a flow swapping scenario in which no other rearrangement is possible and non-synchronous approaches may disrupt existing flows. An example based on [52] is illustrated in Fig. 3.14, where four un-splittable and fixed-bandwidth flows $F_{\{1...4\}}$ traverse switch A. Each link in the network has unit capacity. In case a new flow request F_5 arrives either from D or E to A, F_2 and F_4 need to be swapped to accommodate F_5. In this example, TSO minimize the temporary congestion while not requiring extra network capacity and bandwidth modifications to the existing flows. Simultaneous and synchronous operations are required in the involved switches using time extensions that have been recently standardized in OpenFlow 1.5 by ONF [60] and in NETCONF by IETF [58].

Potential failure scenarios are discussed in [52] which include several switches failing to perform a synchronous operation or controller commands not reaching the destination switches. For these cases, the authors propose the use of TCP as reliable transport protocol for the TSO commands, or simply sending TSO messages sufficiently in advance of the execution time.

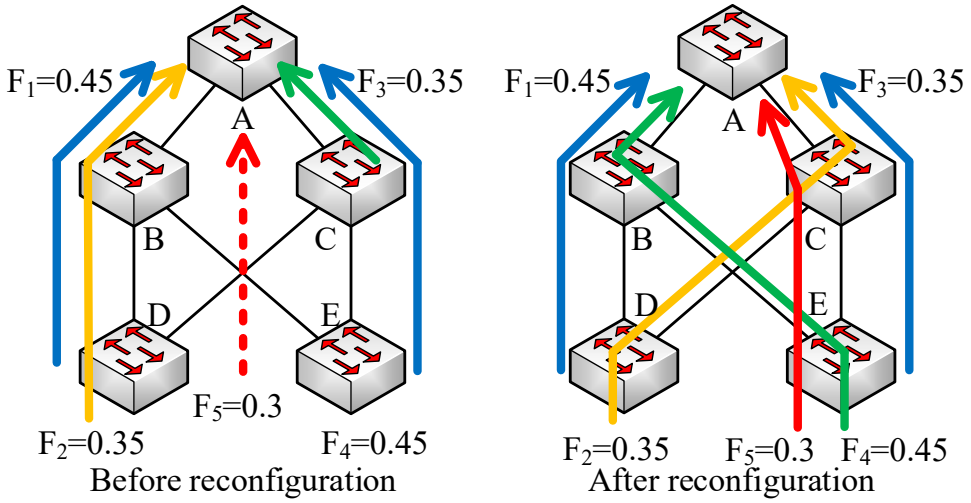


Fig. 3.14: Flow swapping example

TSO in EPN for consistent network updates

Network states evolve with time and it is of paramount importance to keep consistent states between the controller(s) and the network devices, to avoid misbehaviors [67]. Two approaches are commonly used to provide network consistency in the case of state updates. On the one hand, ordered updates are based on sequential operations performed so that no intermediate steps generate network anomalies. This approach requires long reconfiguration times to avoid inconsistency and prevent rapid network updates. On the other hand, two-state updates involve packet tagging by the switches in order to identify whether packets belong to a pre-state or to a post-state update. By doing so, switches are capable to identify which set of packet matching rules need to be applied. The latter approach temporarily requires duplicate rules in the switches until no packet belonging to the pre-state update remains in the network. Thus, extra memory needs to be available in the switches' memory to hold duplicate rules.

Mizrahi et al. [68] address this challenge with a theoretical and experimental analysis using TSO to preserve a given level of consistency during network updates. Their implementation of TSO can be applied to improve the scalability in terms of update duration in both approaches and in terms of extra allocated memory only in the two-state update case. Consequently, a trade-off arises between the desired level of consistency and the achieved scalability. In both approaches, TSO can be scheduled closer in time but at the cost of brief inconsistency periods. For the

two-state approach, limiting the memory resources for the flow table could improve scalability, but increase the inconsistency period.

TSO in EPN for accurate bandwidth monitoring

Megyesi et al. [53] propose the usage of TSO in an SDN-enabled network to improve the measurement of the available bandwidth (ABW). ABW is defined as a dynamic metric to account for the instantaneous amount of traffic that can be added to a path without disrupting other flows. An updated knowledge of the ABW can be exploited by the network operator for agile traffic engineering applications, like highly dynamic routing, traffic consolidation and adaptive video.

Traditional techniques to measure ABW follow two approaches. Active techniques involve pro-actively sending probe packets in the network causing temporary congestion to infer the ABW. Passive techniques may use multiple measurement points in the network and require synchronization of the measurements, thus are rarely used. Authors point out that these techniques are scenario dependent, have limited accuracy and long convergence times.

The major contribution in [53] is an SDN-based ABW measurement application that exploits the global view of the network. An analytical model and an experimental evaluation are reported to address the inaccuracy of the ABW measurements when using the aforementioned application. Inaccuracy occurs because switches are polled by the SDN controller asynchronously and without information regarding the precise sampling time. Subsequently, the authors claim that inaccuracy can be avoided if adopting TSO, and they propose an ad hoc extension of the OpenFlow header to support TSO. In particular, the counter values in the flow tables of the switches are reported to the controller with the corresponding timestamp.

3.6.2 Time synchronized operations in optical networks

In this section we review two recent approaches that exploit TSO in optical networks.

TSO in optical networks for security

Li et al. [54] propose a novel SDN-based security application for optical networks, referred to as fast lightpath hopping (LPH), to prevent eavesdropping and jamming. LPH combines a set of multiple precomputed lightpaths by the SDN controller and TSO among multiple optical nodes. In particular, multi-lightpath computation is performed solving an Integer Linear Programming (ILP) problem, which considers both wavelength and timeslot allocation and minimizes the total number of shared physical links among the lightpaths. Subsequently, the data flow hops among these multiple lightpaths in a sequential manner, as dictated by the SDN controller. TSO, implemented as modifications in the OpenFlow header, enable the synchronization between the involved optical nodes in the LPH procedure. Authors experimentally demonstrate the LPH application in a 4 node testbed, achieving a hop frequency of 1 MHz with acceptable bit error rate.

TSO in elastic optical networks to reduce disruption time for lightpath swapping

This is our work as described in previous sections.

3.6.3 Discussion of TSO in SDN

As noticed in the previous sections, TSO enable a number of novel applications and permit the enhancement of network performance. However, it is important to emphasize that TSO should be considered as a tool which needs to be jointly used with existing techniques to improve performance, to better address existing challenges or to develop new applications. For instance, in EON a challenging scenario occurs when a set of lightpaths present spectrum inter-dependency, preventing parallel defragmentation [69]. Thus, to achieve defragmentation this dependency needs to be broken following a sequence of lightpath rearrangements, which leads to disruption. In this case, TSO can be employed to improve the sequential defragmentation by reducing the disruption time, as discussed in Section 3.2.

3.7 Summary

This chapter reviewed our proposal of TSO in software-defined elastic optical networks. In particular, we employed TSO to minimize disruption time during lightpath reassignment in EON and we discussed the SDN implementation details with NETCONF and OpenFlow exploiting their specific time-extensions. Then, we analytically elaborated that a joint combination of synchronization and bundling operations provides benefits in terms of minimizing the lightpath disruption when swapping is required. Specifically, the TSO-based approaches OF and IA-NC outperform the ASY and N-NC implementations.

Subsequently, we extended the TSO-based proposal with an experimental validation in a five-node metropolitan optical network test-bed. We developed an SDN application that emulates the operations required by the ASY approach to compare its performance against the TSO-based approach. Our reported results validated the convenience of the TSO-based approach against a traditional ASY technique given its reduction of disruption time while both techniques exhibited close network performance indicators (e.g., OSNR, power budget, spectrum tilt) after preforming the lightpath swapping.

Finally, we reviewed recent literature on TSO and discussed its applicability. Regarding electronic packet networks, we highlighted how TSO can improve network performance by efficiently using the available capacity and by managing consistency during network updates. Moreover, TSO can improve the precision of bandwidth monitoring in OpenFlow-based switches. For optical networks, TSO enable lightpath hopping among different routes relevant for a novel security application. Finally, we reviewed our recent proposal that exploits TSO and bundling operations to achieve disruption time reduction during spectrum reallocation.

Chapter 4

Inter-domain and intra-domain network service orchestration

Part of the work presented in this chapter has been published in:

- D. Gkounis, N. Uniyal, A. S. Muqaddas, R. Nejabati and D. Simeonidou, "Demonstration of the 5GUK Exchange: A Lightweight Platform for Dynamic End-to-End Orchestration of Softwarized 5G Networks". In: *44th European Conference on Optical Communication*. Sept 2018, pp. TuDS.14.
- A. Bravalheri, A. S. Muqaddas, N. Uniyal, R. Casellas, R. Nejabati and D. Simeonidou, "VNF Chaining across Multi-PoPs in OSM using Transport API". In: *Optical Fiber Communication Conference*. March 2019. pp. W1G.7.

4.1 Introduction

Due to the upcoming advent of 5G, the demand of network services, having strict requirements such as high bandwidth, ultra-low latency and massive connectivity, is increasing. To sustain these requirements, the network services need to be deployed on a programmable infrastructure to automate and simplify the deployment procedure. This involves utilizing both network and compute resources simultaneously which is known as network service orchestration and is typically performed by a MANO system. Here, each network service (NS) consists of a chain of network

functions; where for deployment agility, these network functions are virtualized, known as VNFs which can run on commodity hardware, as described in more detail in Sec. 4.2.1.

This chapter presents the usage of north-bound interface of an SDN controller to chain VNFs for both inter-domain and intra-domain network orchestration cases. In Sec. 4.2, the 5G UK Exchange (5GUKEx) is presented, which performs network service orchestration among different administrative domains (inter-domain) while using SDN to provision network connectivity between the domains. Sec. 4.3 presents VNF chaining in case of a single administrative domain (intra-domain), where the SDN controller, as part of the WAN Infrastructure Manager (WIM), connects the VNFs running in multiple PoPs over an optical network infrastructure. Finally, Sec. 4.4 summarizes the chapter.

4.2 5G UK Exchange: Light-weight inter-domain network orchestrator

4.2.1 Motivation

5G networks are expected to support a large variety of vertical applications, e.g., smart cities, manufacturing, health, etc., that have diverse and strict requirements. To meet the 5G KPIs such as high bandwidth, ultra-low latency and massive connectivity, there is a need for using compute resources and applying deployment restrictions, e.g., at the edge close to end users, at the core network or even across operator domains. The 5G application demands have made the network operators realize that network technologies and architectures should be programmable and agile to support a wide variety of use cases [70]. Therefore, network operators investigate the adoption of NFV, SDN and MANO as 5G enablers. Network functions run as software on commodity servers in the form of VNFs, facilitating the deployment and scaling while also decreasing operational costs and improving performance. In turn, network equipment can be programmed on demand using SDN, allowing better management of available network resources. The joint coordination of SDN and NFV is assigned to a MANO platform, that performs end-to-end management of network services and infrastructure resources within the various network segments of an operator,

e.g., core, metro, access. This complies with the vision for 5G [70][71] that aims to build a single end-to-end platform over heterogeneous network segments. To this end, organizations such as IETF and ETSI have created standards and guidelines for those systems [72], and open source communities have emerged such as OSM [73], Open Baton [74], SONATA [75] and ONAP [76].

A 5G platform is also anticipated to support multi-operator services which currently experience best effort connectivity and interconnections that fail to consider the service needs. To fulfill the full vision for 5G, orchestration across operator domains should also be performed so services get the required end-to-end Quality of Service (QoS) and can be deployed based on their requirements bypassing any operator boundaries. Collaboration and coordination for end-to-end orchestration among operators can even become desirable from the operators' perspective. Consider, for example, a scenario where one operator does not possess a point of presence in a remote location which is critical to a service with stringent SLAs end-to-end. Upon an agreement with another operator, the end-to-end service can be quickly managed and orchestrated in an automatic way, generating earnings for both parties. Having operators to collaborate combining various available 5G technologies and services, a diverse feature-rich environment can be created that supports innovative and profitable end-to-end 5G services. However, operators would prefer to hide any underlying infrastructure information, e.g., network configurations, that can harm their business and to avoid restrictions on selecting underlying SDN, NFV and other technologies. Another challenge is the lack of a common standardized API among MANO systems that could lead to a "plug-n-play" multi-domain architecture and facilitate the introduction of multiple operator networks.

In this section, we present the 5GUKEx, a novel architecture that aims to enable orchestration of end-to-end network services across multiple administrative domains. In 5GUKEx, the orchestration in each domain is performed by existing MANO systems allowing the operators to be flexible on their underlying implementation and hide any confidential infrastructure information. The MANO systems of each domain connect directly with the 5GUKEx and expose standardized service catalogs [72] which abstract any critical domain information. This allows the 5GUKEx to be a lightweight orchestrator, that sits on top of the MANO systems, to build a common API across the domains and create multi-domain services by combining services offered by the operators. The 5GUKEx performs service brokering, assigning the complex task of resource orchestration to each domain, and dynamic interconnection

of the running services across the domains, keeping the complexity and performance overhead to a minimum. We implement a prototype and evaluate the performance of the 5GUKEx through emulations, showing its lightweight nature.

Organization of the section

The remainder of this section is organized as follows. We first describe the current work in multi-domain orchestration in Sec. 4.2.2. Next, we detail the 5GUKEx architecture in Sec. 4.2.3 and how the 5GUKEx deploys an end-to-end multi-domain service in Sec. 4.2.4. After, we discuss the implementation and performance evaluation of the 5GUKEx in Sec. 4.2.5. Finally we conclude this chapter in Sec. 4.2.6.

4.2.2 Multi-Domain Orchestration: State of the Art

The ETSI NFV standardization group has created the baseline architecture and related standards to enable the development of NFV MANO systems [72]. Most well supported open source MANO systems, such as OSM [73], Open-Baton [74] and SONATA [75], implement the ETSI NFV MANO models for describing the VNFs and Network Services (NSes). However, they are designed to work in a single network domain environment, since there is also a lack of standards that either model the multi-domain NSes or define the interfaces in a multi-MANO communication.

There are also a few works that focus on the area of the multi-domain orchestration for 5G. 5GEx [77] relies on a peer-to-peer interaction of multiple Multi-domain Orchestrators (MdOs), each one administered by an operator, to deploy services end-to-end. Each MdO further interacts with domain orchestrators which consist of SDN or NFV technologies that are responsible for the orchestration of a network segment within an operator domain. X-MANO [78] creates a cross-domain Management and Orchestration platform. The X-MANO architecture introduces Federation Agents (FAs) which provide resource availability in a domain to the Federation Managers (FMs). The FMs can in return work in a peer-to-peer manner with other FMs if needed to orchestrate the network services across multiple FMs.

Both these solutions work in a peer-to-peer manner and they introduce multiple components for both multi-domain service and resource orchestration. In 5GUKEx, we introduce a thin hierarchical multi-domain orchestration layer which builds

on top of existing MANO systems and performs only service orchestration and interconnection, whereas resources are managed and controlled by the individual operators. The proof-of-concept [79] shows the feasibility of this approach.

4.2.3 5GUK Exchange Architecture

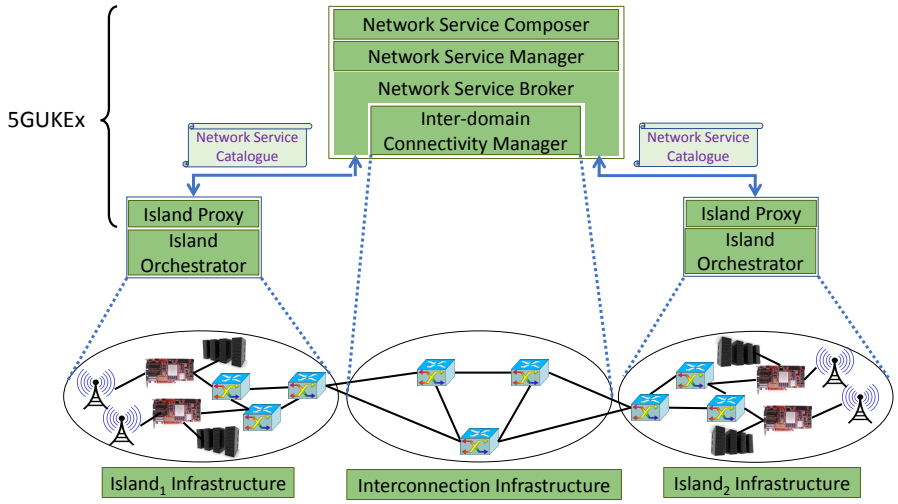


Fig. 4.1: The 5GUK Exchange Architecture

The architecture of the 5GUKEx is illustrated in Fig. 4.1. It assumes that the 5G networks, we call them *Islands*, are individually orchestrated by ETSI-based *Island Orchestrators*, e.g., OSM, OpenBaton, etc., and are connected to the 5GUKEx exposing their network service catalogues. The 5GUKEx is a lightweight hierarchical inter-domain orchestration platform that performs mostly service orchestration. It delegates the heavyweight resource orchestration to the *Island Orchestrators* and interconnects the NSes across the islands, chaining together the running NSes in the individual islands. The 5GUKEx contains multiple components which are detailed as follows.

Island Proxy

It runs on top of the Island Orchestrators and serves as an intermediary between the 5GUKEx and the island orchestrator. The requests from the 5GUKEx to the islands are received by the island proxy and forwarded to the island orchestrator, and

the responses follow the opposite direction. The proxy handles the deployment and termination of running network services on the local islands. Furthermore to the 5GUKEx, it exposes the Network Service Catalogues, i.e., the available network services at the local island in the form of ETSI MANO Network Service Descriptors (NSDs) and optionally, the VNF descriptors (VNFDs), during the registration of an island.

Network Service Broker

It interacts with the Island Proxy, implementing a common API among all the Island Orchestrators, based on the ETSI MANO NSDs/VNFDs and their elements. It receives island registration messages containing NSDs and optionally VNFDs from the Island Proxies to uniquely register the islands and passes the information to the Network Service Manager (NSM). The Network Service Broker is invoked from the NSM during the instantiation, deployment, termination and monitoring processes of network services (NSes) that span across multiple islands, which we call *inter-island NSes*. It receives inter-island NS requests from NSM, identifies and then contacts the relevant islands to perform network service orchestration on each island. Moreover, it aggregates island responses that are part of the same inter-island NS request and forwards them to the NSM.

Inter-Domain Connectivity Manager (IDCM)

This module handles both the control plane connectivity between the islands and the 5GUKEx and the network service connectivity over the *Interconnection Infrastructure* leveraging an SDN controller to program the network elements. A newly registered island to the 5GUKEx provides information to the IDCM about the physical connections of the island to the data plane of the 5GUKEx. The IDCM stores the information and uses it to enable the control plane communication of the 5GUKEx with the Island Proxy. This information is also used during an inter-island service deployment, when the NSM triggers the IDCM passing the network endpoints of a running service on each island. The IDCM combines the network endpoints and the physical connectivity information to enable dynamic network service interconnection among the islands. The IDCM stores the information about the provisioned

service interconnection to be able to terminate it later during an inter-island service termination process.

Network Service Manager (NSM)

The NSM is responsible for the life-cycle management of an inter-island NS. It stores the NS catalogues of the registered islands that the Network Service Composer can access them. It interacts with the Broker for requesting an inter-island NS deployment in the islands and getting island responses about the deployment status and the network endpoints used by the running services which then the IDCM uses to dynamically interconnect the services. Using similar steps of interaction, the NSM can terminate a running inter-island NS.

Network Service Composer

The composer enables users of the 5GUKEx to create inter-island NSes by combining the available NSes of the islands. The composition results in templates of inter-island services that the user can choose to deploy. A deployment request invokes the NSM and further involves a series of steps that is detailed in Section 4.2.4.

4.2.4 Inter-island Network Service Deployment Procedure

As discussed before, the 5GUKEx provides means for the end-user to deploy an end-to-end network service encompassing different operator domains. As shown in Fig. 4.2, this procedure consists of the end-user first selecting different network services of different islands to compose an inter-island NS on the Network Service Composer module on the 5GUKEx. Then, the end-user requests the instantiation of the inter-island service, triggering the 5GUKEx to instantiate on the corresponding islands the individual network services which are part of the composed inter-island NS. The islands verify if they have enough local resources, e.g., compute, storage and memory, to deploy and start the individual network service. If there are available resources, each island creates a Network Service Record (NSR) and then sends the information to the 5GUKEx to signal that the request can be fulfilled. When the 5GUKEx receives the responses from all the relevant islands, it informs the user that

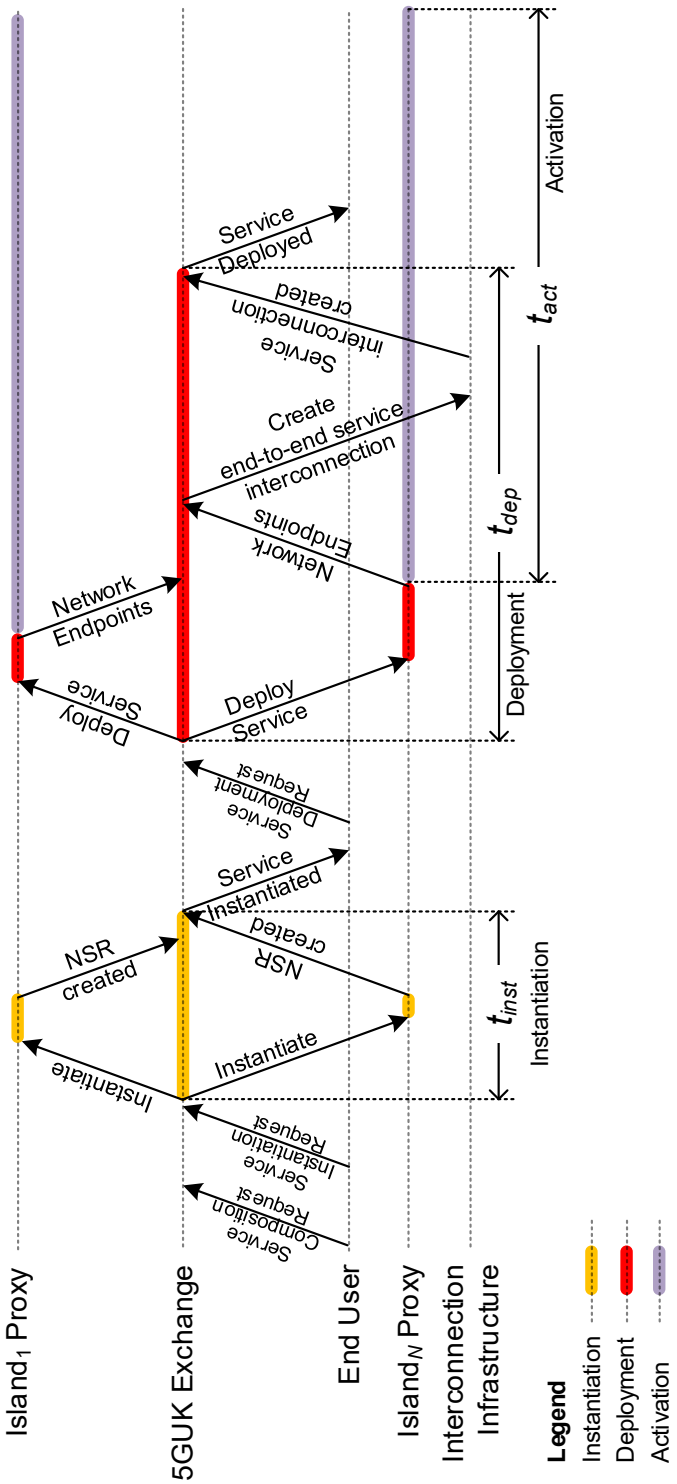


Fig. 4.2: Network Service Deployment Procedure in the 5GUK Exchange

the islands are ready to deploy the inter-island NS end-to-end. The instantiation time is referred as t_{inst}.

Once the user chooses to deploy the inter-island NS, the 5GUKEx contacts the islands to deploy the previously instantiated NSR. Upon receiving the deployment message by the 5GUKEx, the local island proceeds to deploy the network service using the island orchestrator. Once the service is being activated, the local island provides information to the 5GUKEx about the network endpoints to be used at the island gateway. Once the endpoints are received from all the islands, the IDCM module of the 5GUKEx creates the underlying data plane network service interconnection across the islands. After the inter-island service interconnection is provisioned, the deployment procedure is finished and the end-user is notified. The time for deploying an inter-island service is referred as t_{dep}. Meanwhile, the network service activation is carried out by the local islands and takes t_{act} time i.e., from the time that a service is deployed until it becomes active.

4.2.5 Implementation and Performance Evaluation

We have built the 5GUKEx in Python, based on the 5GUKEx architecture shown in Fig. 4.1. To enable the IDCM to dynamically control the interconnection infrastructure among the islands, we have used the ODL SDN controller. We have also built the common inter-domain API of the 5GUKEx considering OSM as the Island Orchestrator at each island.

Setup

To evaluate the performance of the 5GUKEx, we emulate the 5GUKEx and 4 islands. We use 5 Dell PowerEdge T360 servers, each equipped with Intel Xeon E5-2680 CPU with 56 cores and 64GB of RAM running Ubuntu 16.04 as operating system. One server hosts the 5GUKEx and each of the remaining servers emulates a local island; each island consists of the ETSI NFV-compliant OSM as the local island orchestrator and OpenStack for management of the compute resources. Each server also hosts an instance of the ODL controller which controls the network resources of each island. We use two Corsa DP2100 OpenFlow switches [80] for the network data planes of the islands and the 5GUKEx, which are interconnected. To emulate the network resources of the islands, a Corsa SDN switch is shared among the islands

by reserving dedicated ports per island. For the interconnection infrastructure of the 5GUKEx, the second Corsa switch is used. All the islands have identical server, switch port and switch bridge configurations.

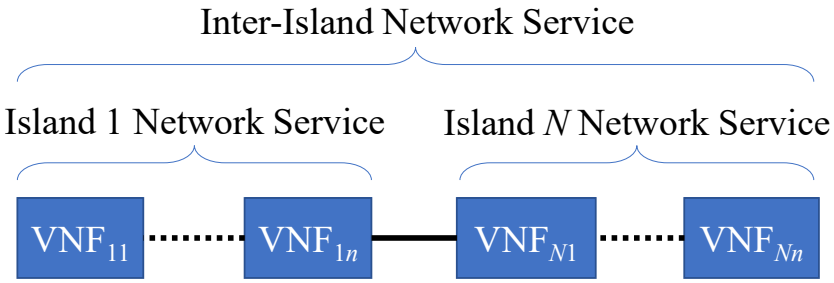


Fig. 4.3: VNF chain spanning multiple islands

Experiment

We use the 5GUKEx to instantiate and deploy inter-island network services across multiple islands and measure the instantiation t_{inst}, deployment t_{dep} and activation t_{act} times. Each island exposes the same NSDs and VNFDs that the user composes to form an inter-island service by selecting a number of NSDs from multiple islands. A network service at each island consists of n connected VNFs and the network services of the islands are stitched to each other by the inter-island network infrastructure using L2 connectivity to create an inter-island network service. This results in an inter-island network service that chains together multiple VNFs across multiple islands as shown in Fig. 4.3. Each VNF consists of a CirrOS [81] which is a minimal Linux image used here as a baseline VNF.

To evaluate the performance of the 5GUKEx and its promise for lightweight inter-domain orchestration, we perform two measurement campaigns. Since the 5GUKEx delegates the resource orchestration to the local islands, we first measure the local island orchestration time (t_{act}) and then we measure and compare with the inter-island coordination and interconnection times.

Results

We first deploy an NS consisting of n connected CirrOS VNFs at each local island without using the 5GUKEx to measure the local island network service activation

time t_{act}. We run the tests for 20 times and the results are shown in Table 4.1 with 95% confidence. The time taken for a service to be active is at least 27.10 seconds for the case of a network service consisting of one VNF per island.

Table 4.1: Network Service Activation Time at each Island

Number of VNFs n per Island	**Time till activation [sec]**
1	27.10 ± 1.04
2	43.25 ± 1.40
3	64.66 ± 2.24
4	89.59 ± 1.56

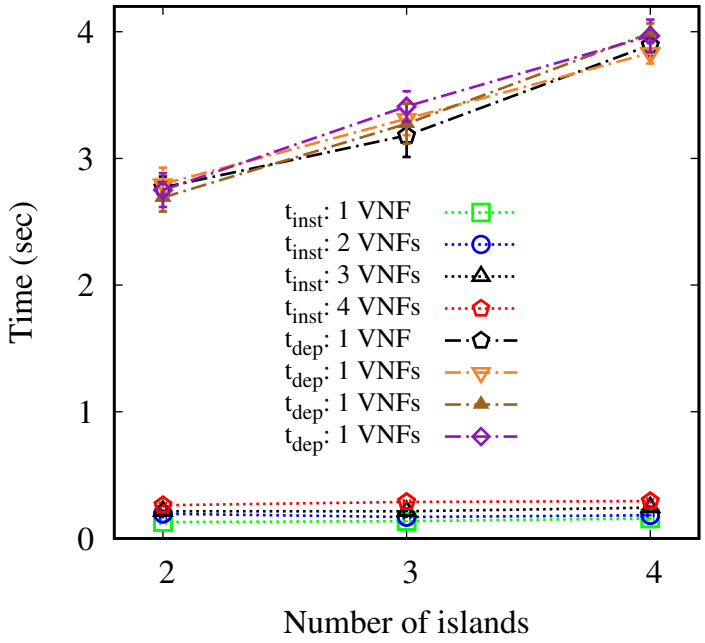


Fig. 4.4: 5GUKEx Instantiation and Deployment Times

We then deploy from the 5GUKEx an inter-island network service that consists of the same network service as before, now deployed at multiple islands. We run sets of 20 tests, modifying the number of VNFs per NS and the number of islands used on the inter-island service in each set; and we measure the instantiation t_{inst} and deployment t_{dep} times along with their 95% confidence intervals, as shown in Fig. 4.4. In contrast to the activation time t_{act}, the t_{inst} and t_{dep} times are minimal. The t_{act} for a network service containing four VNFs per island requires 89.59 seconds,

whereas the deployment from the 5GUKEx even across four islands takes only about 4 seconds. This is due to the fact that the 5GUKEx performs service orchestration and minimal network (re)configurations of the inter-island network infrastructure and because it delegates the performance-heavy orchestration of computational resources on the local islands. This shows that the 5GUKEx is a thin layer of orchestration with minimal overhead, consequently contributing towards the sustainability of the 5GUKEx platform. We note that during the current tests, we consider that the islands have already chosen the network endpoints to use at the gateways of each island, before the service deployment from the 5GUKEx, to pass the relevant traffic across their domains. These endpoints are thus only exposed to the 5GUKEx during the deployment phase to pass them to the IDCM module to interconnect the traffic. We could also allow dynamic creation of network endpoints to be communicated to the 5GUKEx after the service activation at each island but this would have not affected much the results, since the service activation is a performance-heavy operation as shown before.

Regarding the deployment time t_{dep}, we see that it increases when network services are deployed across more islands. This is due to the fact that more network flows are installed on the inter-island network switch to steer the relevant traffic accordingly. Furthermore, we see that t_{dep} remains the same with increasing number of VNFs per NS deployed at an island. This is due to the nature of the 5GUKEx being a thin orchestration layer as described earlier. In addition, the instantiation time t_{inst} is negligible compared to the deployment time t_{dep}. By adding the t_{dep} and t_{inst} times and comparing them to the t_{act} at each island, we can see that the total inter-domain orchestration overhead is minimal.

4.2.6 Conclusions and Future Work

In this section, we presented the 5GUK Exchange, a hierarchical multi-operator platform that aims to orchestrate end-to-end network services in a sustainable manner. The 5GUKEx builds a multi-operator API that is based on standards, allowing operators to integrate using their existing MANO systems, to hide any confidential infrastructure information and to provide flexibility in selecting any underlying SDN, NFV technologies. By brokering the orchestration of the individual network services to operators, the 5GUKEx becomes a a lightweight solution in performance and complexity that performs multi-operator coordination and service interconnection.

We presented the architecture of 5GUKEx; we implemented a prototype based on the service catalogues of the ETSI NFV MANO-compliant OSM and we evaluated its performance showing that the multi-operator orchestration layer of the 5GUKEx has minimum operational overhead. As a future work, we plan to integrate to the 5GUKEx NFV MANO systems that are based on TOSCA service catalogues, such as ONAP. We also plan to introduce layer-3 connectivity in our 5GUKEx deployment, allowing additional testbeds to integrate and thus enabling the 5GUKEx to become a solution towards federating 5G testbeds worldwide. Furthermore, we plan to carry out more experiments to compare the performance of 5GUKEx with other 5G orchestrators.

Acknowledgment

This work has received funding from the UK in the context of DCMS UK 5G Testbeds & Trials Programme and EPSRC projects TOUCAN (EP/L020009/1) and INITIATE (EP/P003974/1) and from the EU in the context of the H2020 projects Metro-Haul, MATILDA and 5GinFIRE (grant agreements 761727, 761898 and 732497).

4.3 VNF Chaining across Multi-PoPs in OSM using Transport API

4.3.1 Introduction

There has been recent shift in telecommunications industry towards technologies like NFV and SDN as mentioned in Sec. 4.2.1. The hardware network equipment is being replaced by VNFs, which can run in remote datacenters as well as the edge computing nodes. The constraints in the placement of VNFs within NFV [82] is resulting in an increase of inter-datacenter traffic, which is forecasted to reach up to 14% of the total data-center traffic by 2021 [83]. The bandwidth and latency sensitive use-cases like Augmented/Virtual Reality, 8K 360° video transmission etc., where some processing is done in remote datacenters and other on the edge nodes, will be some of the contributors to this increase in traffic. The distribution of VNFs across remote and

edge-computing nodes for realizing these use-cases has made it necessary to create an end-to-end network and resource control. It has also created the need to build the underlying networks to satisfy the stringent requirements of low latency and high bandwidth across the cloud and edge computing infrastructure. In order to achieve these targets, projects like Metro-Haul [84] are creating an ecosystem to provide high bandwidth, low latency, end-to-end connected and programmable networks. Such stringent network requirements can only be sustained using optical networks in the long run [85]. Furthermore, Metro-Haul utilizes ETSI NFV MANO systems to orchestrate network services over optical metro networks, to create end-to-end network services.

ETSI NFV MANO standards include the Wide-Area Networking (WAN) Infrastructure Manager (WIM) which allows to orchestrate network services over multiple data-centers in a WAN. However, due to the heterogeneous nature of the underlying networks aligned with the complexity of end-to-end orchestration, its functionality and APIs have not been clearly defined yet [86]. In this case, a Transport API (T-API) [87] based WIM can be used to integrate optical networks within an NFV MANO system, for network services with VNFs having high-bandwidth and low latency requirements. T-API has been standardized within the ONF by the Open Transport Configuration & Control (OTCC) group and is being increasingly adopted by the industry. Thus, using T-API for integration of optical networks with the existing NFV-MANO systems allows easier upgrade and operation of existing MANO systems. Furthermore, T-API simplifies the interdomain networking, encompassing both electrical domain and optical domains and abstracts the complex underlying topology.

In this section, we propose and implement an architecture to integrate T-API with the leading ETSI NFV MANO system i.e. OSM [73] for the first time, which allows orchestration of VNFs across data-centers over the underlying optical networks. The whole system including the data plane is demonstrated where the applicability of T-API as interface towards the WIM is established, using SDN controllers for multi-domain network orchestration.

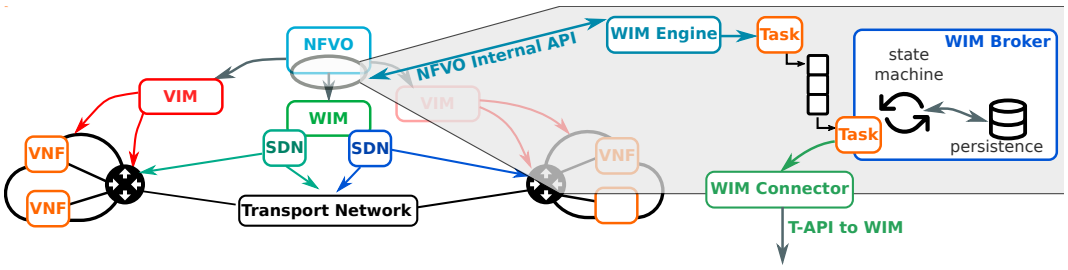


Fig. 4.5: Proposed WIM Architecture in the context of ETSI NFV MANO framework [90]

4.3.2 T-API as WIM North-bound API

The choice of T-API as the interface between the OSM and the WIM is motivated by multiple factors. T-API, as defined by the ONF, has fulfilled its goal of having a common standardized North Bound Interface (NBI) across multiple network controllers, to be used e.g., by a network orchestrator or applications. It builds on core models that are technology agnostic (aligned with the ONF Core Information Model, which defines a common object model for all types of Software Defined Networks, including components like network resources or service constructs) while allowing technology-specific extensions for relevant layers. T-API follows a model driven development: models are defined in UML and automatically translated into YANG which, in turn, may be automatically validated and stubs generated for common programming languages. From the point of view of maturity and adoption, it has been demonstrated in multiple proof-of-concepts, interop events and is supported by many vendors [88]. It has been adopted by several initiatives and SDOs, such as MEF OpenCS Optical Transport or, for the latest v2.1 release, the ODTN Project [89] as the NBI for a controller of an optical disaggregated transport network. Release 2.0 added the ability to take into account node constraints, protection services and consistent OAM and monitoring, and the latest 2.1 release includes new models for the photonic layer, with support for flexi-grid network media channels, including model for the OCH, OTSi, OTSiA, OTSiG, OMS, OTS and Media channels as per ITU-T G.872 (2017) version 4.

4.3.3 Architecture for WIM Integration in NFV MANO

As shown in Fig 4.5, our proposed architecture conforms to the ETSI NFV MANO standards [90] focusing on the orchestration on VNFs across multiple data-centers (PoPs). A request is issued to the NFVO to deploy a NS composed of VNFs in a multi-PoP environment where the location of each VNF corresponding to the PoP is specified. With this information, the NFVO requests each underlying Virtual Infrastructure Manager (VIM) to instantiate the corresponding VNFs and gets information about the related local virtual networks where they would be attached. Meanwhile, the NFVO assigns to one or more WIMs the responsibility to create WAN links by configuring the underlying transport network via T-API.

In order to realize the aforementioned steps, we propose the introduction of sub-components inside the NFVO, as indicated in Fig. 4.5. This architecture is based in the execution of asynchronous tasks to avoid blocking behaviour and improve the overall network service provisioning time. The **WIM Engine** has two responsibilities: firstly, to provide information to the NFVO about the available WIMs and the characteristics of the connectivity they are able to sustain; and secondly to specify and schedule a series of tasks that encode the instructions for establishing WAN links. While each task works as a standalone processing unit, implementing a state machine upon activation, the **WIM Broker** coordinates their threaded execution in a task queue. By tapping into a shared communication mechanism internal to the NFVO (implemented via either a common database or event stream), tasks can verify preconditions and fire actions to the external WIMs accordingly. As an example, special information, such as VNF IP and VLAN segmentation id, might need to be published by a VIM to the NFVO after its workload is processed. To wait for this information, the tasks can be suspended, and rescheduling them is the responsibility of the WIM Broker. Conversely, when all the preconditions are met, a task invokes commands on a **WIM Connector**. Although the WIM Connector is designed to abstract the WIM API permitting protocol-agnostic systems, in this chapter, we propose the usage of the T-API due to its support to a wide variety of transport technologies and SDN topologies.

4.3.4 Implementation and Experimental Demonstration

We have developed a preliminary implementation of the proposed architecture using OSM as baseline system to demonstrate and evaluate the use of T-API with MANO for NFV orchestration over a Multi-PoP infrastructure. This implementation introduces a new WIM subsystem in OSM Resource Orchestrator (RO), containing the sub-components described in Section 4.3.3. For demonstrating its applicability, a network service composed by 3 VNFs (Fig. 4.6) was deployed over an an optical infrastructure. Fig. 4.7 shows the experimental setup, which consists of two OpenStack datacenters interconnected by a combination of packet switches and optical cross connects (OXCs) which are SDN-enabled and managed by a combination of SDN controllers. On top of the SDN controller, we employ a T-API proxy based on a T-API 2.0 reference implementation [91]. The T-API proxy exposes two service endpoints (*svc_ep_1* and *svc_ep_2*), at both edges of the interconnection network between the datacenters, to OSM. The information about these service endpoints is added to OSM before network service deployment as part of a port mapping process, making them available for usage.

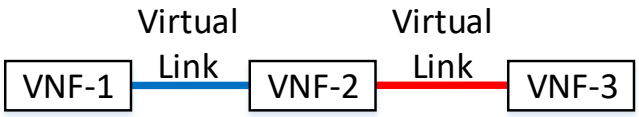


Fig. 4.6: Network Service

At the time of deployment of the network service by OSM, the location of VNFs is specified, such that the network service spans across multiple datacenters as in Fig. 4.7. This is followed by OSM deploying the VNFs at the specified datacenters through the VIMs. In addition to the VNFs, the VIMs also create the network connected to VNF-2 and VNF-3, which spans outside the datacenter to the transport network with VLAN *A* and VLAN *B* respectively. Once the VNFs and the networks are created, the VIM sends the VLAN information to the OSM. OSM detects that the virtual link connecting VNFs 2 and 3 spans multiple datacenters. Once OSM receives the VLAN information, it uses the pre-defined port mapping to obtain the T-API service endpoints connected to each datacenter. The T-API service endpoint (*svc_ep*) and the VLAN for each datacenter is sent to the T-API proxy to provision a network between the datacenters. The T-API proxy translates this information into low-level device specific flow installation using the SDN controller to interconnect

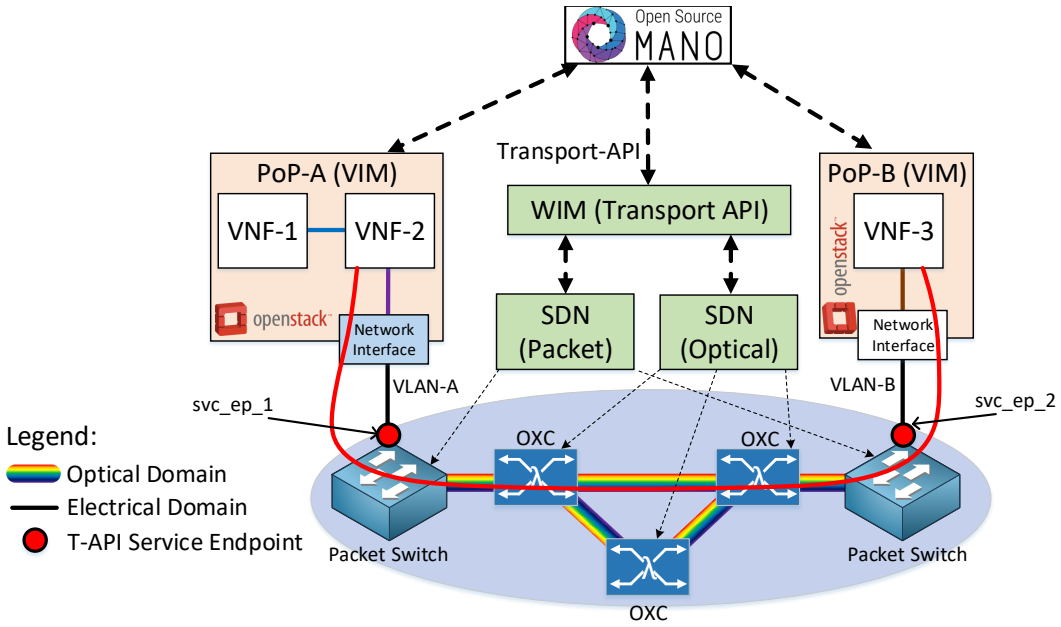


Fig. 4.7: Testbed for T-API with OSM

the two VLANs at the either datacenter, as shown by the red line in Fig. 4.7. Fig. 4.8 shows the Wireshark capture of the T-API *Create Connectivity Service* message that OSM sends to the T-API proxy during network deployment. It includes the two T-API service endpoints and the VLANs used at either datacenter, which is used to provision the end-to-end network between the datacenters.

4.3.5 Conclusion and Future Work

In this section, we presented the integration of standard T-API with OSM, a popular ETSI based open source MANO system, focusing on the orchestration of VNFs over multiple PoPs connected over optical networks. The proposed architecture was validated and demonstrated using a multi-datacenter and multi-domain network experiment.

Our presented implementation is part of the OSM Release 5. In future, the upcoming T-API 2.1 photonic extensions will be exploited to improve optical network orchestration capabilities.

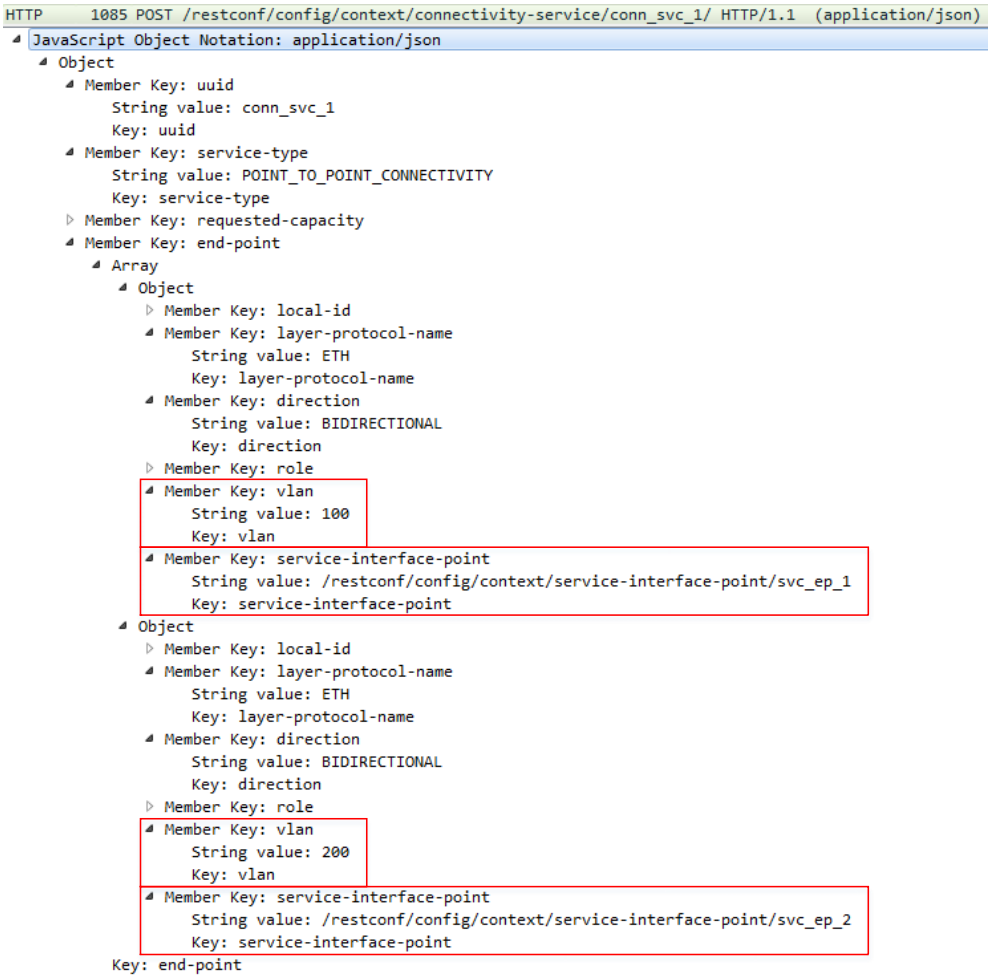


Fig. 4.8: Wireshark Capture along with Create Connectivity JSON

Acknowledgment

This work has received funding from EPSRC grant TOUCAN (EP/L020009/1) and EU H2020 Metro-Haul (grant agreements 761727).

4.4 Summary

This chapter presented an application of the North-bound interface of the SDN controller: inter-domain as well as intra domain network service orchestration. In

both cases, the SDN controller is used to create networks to chain VNFs which are part of a network service.

In case of the inter-domain network orchestration, we presented the 5G UK Exchange, which is a light-weight network service orchestrator to deploy inter-domain network services while offloading the computationally intensive resource allocation to the individual domains. As shown by results, the instantiation and deployment times at the 5GUKEx are minimal as compared to the actual activation time of a network service; this depicts the light-weight nature of 5GUKEx. The IDCM part of the 5G UK Exchange includes the SDN controller to chain VNFs hosted at PoPs at different domains.

In case of the intra-domain network orchestration, we extended OSM to interact with the WIM using Transport-API to orchestrate network services with the connectivity between the VNFs, hosted at different PoPs, using an optical network based infrastructure. The WIM uses Transport-API as the North-bound interface and uses the North-bound interface of the individual SDN controllers for both packet and optical domains to orchestrate an end-to-end service between different PoPs. We conduct an experimental demonstration of our system where the implemented WIM connector is an open source contribution to OSM Release 5.

Part II

Stateful Data Planes

Chapter 5

Replicated states in stateful data planes

5.1 Introduction

In recent years a major shift of paradigm has been observed in SDN with the introduction of stateful data planes. This interest arose due to the fact that stateful data planes carry a great potential in filling the performance gaps which arise due to the complete centralization of control plane in SDN [1][92]. The key enablers of a programmable data plane are:

- Programmable pipelines which allow to execute arbitrary user-defined code during packet processing
- Introduction of persistent memories which allow to maintain persistent states inside switches. These may also be referred to as state variables.

The combination of these two elements provides an additional level of programmability with respect to traditional SDN and a way of defining complex in-network processing rules. Consequently, the data plane switches can take some decisions locally without relying on the intervention of an SDN controller [93]. This greatly improves the reactivity for the vast majority of network applications by removing latency overhead, previously caused by the interaction with the controller; consequently minimizing the computational burden the controller must undergo in order

to sustain the correct network behavior [94]. At the same time, the presence of persistent memories in the form of state variables enables new fine-grained networking applications [95] as decisions can now be taken on a per-packet basis contrary to the per-flow basis which was present in legacy SDN [2].

Following this paradigm shift, numerous works have focused on trying to explore the novel capabilities provided by stateful data planes by defining frameworks with the goal of providing deeper network programmability. As an example, authors of SNAP [11] show that it is possible to implement complex network policies distributed in the form of *State Variables*, *States* for simplicity, across multiple devices; and define complex per-packet processing rules. SNAP however neglects numerous aspects related to the scalability and performance issues which inevitably arise when dealing with network-wide policies. In particular, the authors assume that any given network policy is constrained to have a unique single copy of a particular state variable inside the entire network, which forces all flows necessitating that particular policy to traverse a given set of switches. This inevitably leads to the creation of bottlenecks and consequently performance degradation for flows satisfying that particular policy. At the same time, in case of network-critical policies, failure of even a single device holding the state will eventually jeopardize the integrity of the entire policy and may lead to the irrecoverable loss of persistent policy-related states.

In order to overcome these issues, in this thesis, we extend previous proposals by allowing synchronized policy replication among different network devices. Our proposal directly addresses the limitations related to scalability and fault-tolerance by introducing *replicated state variables* across the network. This allows to have multiple distinct copies of the same state variable inside the network thus providing opportunities for load balancing. At the same time, although being physically distributed, thanks to the presence of a synchronization mechanism, different state variable replicas are allowed to behave as a single replica by sharing common states among them. Our initial proof of concept [96] shows that the concept of replicated states can be implemented with state of the art programmable data planes such as P4 and OPP. Following these results, we question what is the optimal trade-off between network resource utilization and network performance. In order to answer this, we address optimal placement of states in the network in this chapter. The problem targets the performance of flows by minimizing the amount of traffic in the network which consists of both data traffic and synchronization traffic among the replicas. We formalize the problem in the form of a ILP formulation. The results show that in

the case of multiple replicas, it is possible to achieve significantly lower overhead in terms of network resource utilization as compared to the case of a single state. Following these results, due to the extreme computational complexity of the ILP formulation, we present a heuristic for the placement of replicated states. We indicate that even a simple heuristic based on clustering is able to achieve near-optimal results.

5.1.1 Organization of the chapter

The remainder of this chapter is organized as follows. In Sec. 5.2, we further elaborate the stateful data planes and state replication. In Sec. 5.3, we present the formalization of the optimal state replication problem for multiple replicas of the states. In Sec. 5.4, we propose an approximated algorithm for the single state replication problem and we show performance trade-offs between data and synchronization traffic depending on the number of copies of the state. In Sec. 5.5, we present an asymptotic analysis to compute the required number of state replicas in a network. In Sec. 5.6 we discuss the related works. Finally, we summarize in Sec. 5.7.

5.2 State replication in stateful SDN

5.2.1 Stateful data planes

Stateful data planes [6][7] introduce the possibility for switches to store persistent states related to network dynamics. In additional to the introduction of persistent memories, differently from traditional SDN data planes, switches are also equipped with programmable logic units which allow network operators to define different per-packet processing policies. The consequence of this innovation is that switches are now provided with the possibility of performing local decisions based on their internal states without any interaction with the SDN controller. This enhancement opens a wider field of opportunities in respect to standard SDN paradigms such as the ones based on OpenFlow. Notably new opportunities arise for the definition of more fine-grained processing policies since by reducing the interaction with the controller, switches are able to capture network events on a per-packet granularity.

5.2.2 SNAP programming abstraction

Following the increasing need for highly dynamic network services and policies, multiple frameworks aiming at providing tight control over the way traffic flows are processed have been proposed. With the introduction of programmable data planes, numerous traffic processing policies have been pushed directly to the switches. There has been observed a substantial increase in the complexity of the required frameworks since resources required for traffic processing are made available across multiple devices and in much limited quantity. Following this shift of paradigm, new frameworks able to embed user-defined network programs to the switches have been proposed [11][97]. Among those frameworks, the most relevant and versatile is SNAP [11].

In order to cope up with the increase in complexity introduced by the presence of sparse computational resources across the network, SNAP introduces a one-big-switch (OBS) model as an abstraction. With such abstraction, the whole network of switches is seen as a single big switch with a given set of input and output ports and an aggregate list of available resources for traffic processing. Due to the way the OBS abstraction is defined, flow routing is described on the basis of I/O port pairs.

When developing a network program or policy, the programmer will be exposed to a single big switching fabric with a given computational capabilities without having any knowledge of the actual underlying composition of the network. Instead of designing network policies from scratch by embedding elements of the policy on each single device, SNAP provides the programmer a specific programming language. This language, which is an extension of NetCore [98], allows to express most of the network functions by means of algebraic expressions and, most importantly, can be interpreted by SNAP. All network programs are decomposed by SNAP into a xFDD, an extension of forward decision diagram which incorporates also stateful elements. xFFD alongside with traffic requirements are fed into SNAP ILP (Integer Linear Programming) optimizer which then performs embedding of each single element of the decomposed policy inside single switches and the consequent routing of the traffic flows requiring particular network policy through switches storing it.

The embedding process embeds both processing logic and stateful elements, namely state variables. Consequently the routing is chosen to guarantee that flows traverse all switches where the involved states are located. The order in which traffic

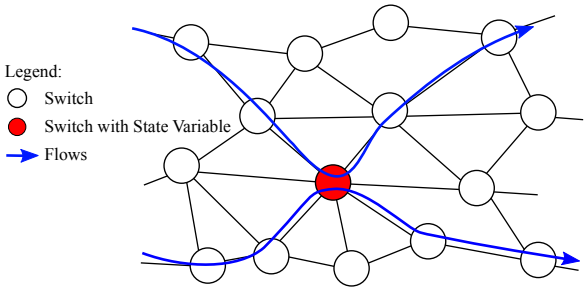
traverses the state-holding switches pays a fundamental role as state dependencies must be preserved in order to correctly reconstruct the xFDD of the original policy. Thus, internally the routing decisions do not generally follow the shortest path between the input and output port in the OBS. Yet the SNAP solver still tries to minimize the total data traffic in the network.

The main limitation of SNAP emerges from the fact that it allows only one unique position per state variable inside the network. This considerably restrains the routing of flows and consequently precludes a wide range of optimization techniques such as load balancing and traffic engineering. Indeed, flows affecting or affected by a state must be detoured from their shortest path routing in order to traverse the state-holding switch, ultimately leading to a non-optimal network resource utilization.

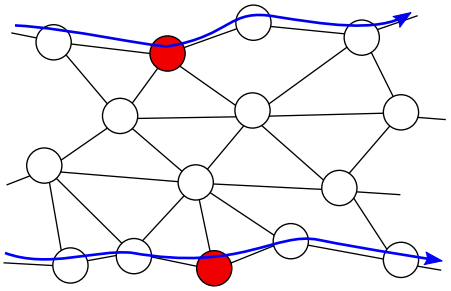
5.2.3 State replication

In order to cope up with the limitations of SNAP, we propose to relax the condition which imposes the uniqueness for each state variable by allowing to replicate state variables inside the network. In our work we address the optimal placement of the copies of state variables across the network, given the knowledge of the traffic demands and of the xFDD, as computed by SNAP compiler. These modifications are made uniquely at the level of ILP optimizer and do not impact any other functionality of SNAP. In this way we provide the possibility of having multiple copies of the same state variable distributed across multiple switches, which reduces the overall traffic in the network and provides load balancing across multiple state variable copies.

As an example, consider a global counter that must be updated by all flows and depending on its value, adopt an action targeting all flows. SNAP will place a single copy of the state variable in a single switch in the topology, likely the switch in the most “central” position in respect to the network topology as shown in Fig 5.1a, where the most “central” position refers to a node in a graph with the highest betweenness centrality. As a consequence, all flows will be forced to be routed through the only state holding switch and then each forwarded to the corresponding destination port. In a scenario with subset of flows with a sufficiently large demand in terms of bandwidth, the ILP solver of SNAP would provide either an unfeasible solution or would not be able to meet the bandwidth demand of the flows. With our proposed solution instead, the ILP optimizer will replicate the state



(a) State placement using SNAP



(b) Replicated States using our solution

Fig. 5.1: State placement in the network

variables across multiple devices leading to a better balance of traffic inside the network as shown in Fig 5.1b. We claim that by permitting few copies of a state and carefully distributing them in the network, i.e., a limited level of state replication, the congestion due to the legacy SNAP approach can be heavily reduced. This comes at some cost, since copies of the same state must be synchronized by means of an appropriate mechanism. However, even in the presence of synchronization traffic, the reduction in the overall traffic overhead in respect to the traditional SNAP approach remains significant as shown in the results in Sec. 5.4.1.

The choice of an appropriate synchronization mechanism, i.e. of a replication scheme represents a critical aspect for what concerns the performance of the system and its implementation complexity.

The CAP theorem [17] states that for a replication scheme out of Consistency, Availability and Partition tolerance, only two properties can be picked at the same time. Considering that network failures may occur, partition tolerance cannot be left out of the design of our replication algorithm, leaving us with two main reference models: *Strong Consistency* and *Eventual Consistency*.

Strong consistency A replication algorithm based on strong consistency privileges consistency over availability. This translates into strong guarantees for the values of replicated items stored at each replica at the cost of big delays during the manipulation of those items. The increase in latency is caused by the presence of a complex protocol [19][99] requiring intensive interaction among the replicas whenever a transaction involving replicated items must be executed. A side effect of the presence of such complicated replication protocols is a considerable increase in the synchronization traffic overhead and significant increase in the implementation complexity.

Eventual consistency On the other hand replication schemes based on eventual consistency prioritize the availability of the replicated items over the consistency. This translates into low latencies during the execution of transactions at the cost of no guarantees over the actual value of replicated items inside each replica. Most of eventual consistency algorithms are based on gossiping and its variants [100–102] which incur into small overhead due to the synchronization traffic. At the same time due to the simplicity of the communication protocol these kind of algorithms can be easily implemented inside devices providing a limited amount of hardware resources.

For the particular case considered in this thesis, replicas are composed of switches with limited amount of hardware resources, thus incapable of executing complex algorithms. At the same time, the arrival rate of transactions is comparable with the aggregate arrival rate of packets at any given switch. These considerations lead to significant issues in implementing a replication algorithm employing strong consistency. Indeed, the latency given by communication delays among replicas would require to buffer packets at each switch while waiting for the outcome of the replication transaction.

For this reason we consider a replication scheme based on eventual consistency among the replicas of the same state, thus disregarding consistency property in change of availability and simpler implementation. As already shown in a PoC implementation [96], this choice leads to low complexity while at the same time is able to maintain small replication error among replicas.

5.3 Optimal state replication problem

The objective of the state replication problem is to identify the best nodes where to place the copies of the state and compute the optimal routing, in order to minimize the traffic on the network and guarantee that all the flow affecting one state variable will traverse at least one copy with this state. The traffic on the network is not just due to the data, but also the synchronization to keep consistent copies of the same state.

We propose a integer linear program (ILP) formalization, with the following input parameters, as in the original SNAP model. The relevant notation is reported in Table 5.1.

- *Network.* Let $G = (V,E)$ be the network graph with N nodes. Let c_e be the capacity of edge $e \in E$.
- *Traffic flows.* Let $\mathscr{F}$ be the set of all the flows. Coherently with [11], we assume to know the traffic demands in advance: let λ_f be the demand of traffic flow $f \in \mathscr{F}$, with $f_s \in V$ as source node and $f_d \in V$ as destination node.
- *State variables.* SNAP compiler, based on the operating network applications, identifies the sequence of state variables to traverse for each flow, assumed to be known in advanced coherently with [11]. Let S be the set of all state variables. Let $S_f \subseteq S$ be the *ordered* sequence of state variables for flow $f \in \mathscr{F}$.
- *Maximum number of copies.* Let $C_s \geq 1$ be a given upper bound on the number of copies for a state variable s, chosen by the network designer. The actual chosen number of copies while minimizing the state synchronization traffic is $\hat{C}_s \leq C_s$

Let H_f consist of all possible sequences of state copies for a flow f. E.g., if a flow f requires 3 state variables with at most 2 copies each, then $H_f = \{[111],[112],[121],[122],[211],[212],[221],[222]\}$. Consider now one sequence $h \in H_f$; let h_s be the specific index of the copy of state $s \in S$ appearing in h. E.g., assume $S_f = \{A,B,C\}$ and $h = [122]$. Then h_A would be the copy 1 of state A, h_B would be copy 2 of state B and h_C copy 2 of state C.

The main output of the solver is described as follows, and the relevant notation is reported in Table 5.2:

- *Placement of the copies of each state.* Let P_{scn} be a binary variable equal to 1 iff copy c of state s is stored at node n. Note that the optimization problem might place multiple copies on the same node, but this would correspond to a single instance of the state. Thus, the actual number of copies $\hat{C}_s$ of state s across the whole network can be computed as follows[1]:

$$\hat{C}_s = \sum_{n \in V} \mathbb{1}_{\{\sum_{c \leq C_s} P_{scn} > 0\}}$$

- *Data traffic routing.* Let R_{fhe} be a binary variable equal to 1 iff flow f traverses the sequence of states copies h on edge e. The set of such variables describes completely the routing of all the flows across the network, taking into account the constraint for the required sequence of traversed copies. To avoid out-of-sequence problems, we do not allow flow splitting between different sequences of copies.

- *Synchronization traffic routing.* Let $\hat{R}_{snme}$ be a binary variable equal to 1 iff there are copies of the state variable s on nodes n and m and the flow from node n to node m traverses edge e. The set of such variables describes the routing of the synchronization traffic between the different copies of the same state. Let $\hat{\lambda}_s$ be the traffic generated by each copy of the state to update the other copies.

In the optimal state replication problem, one possible objective function is to minimize the total traffic in the whole network, as considered in SNAP [11]:

$$\min \sum_{e \in E} \sum_{f \in \mathcal{F}} \sum_{h \in H_f} R_{fhe} \lambda_f + \sum_{e \in E} \sum_{s \in S} \sum_{n \in V} \sum_{\substack{m \in V \\ n \neq m}} \hat{R}_{snme} \hat{\lambda}_s \tag{5.1}$$

The first term is the data traffic, obtained by summing all the traffic due to f on all the possible sequences of state copies and on all the edges. Instead, the second term is the synchronization traffic between any copy of the same state, summed across all the states and edges in the graph. Notably, (5.1) corresponds to the same objective

[1] Let $\mathbb{1}_{\{A\}}$ be the indicator function of A, equal to 1 iff condition A is true.

Table 5.1: Input Parameters/Sets

Variable	Description	Range
G	network graph	
V	set of all nodes	
E	set of all edges	
N	number of nodes	$1,\ldots,\lvert V\rvert$
$\mathscr{F}$	set of all the flows	
f_s	source node for flow f	$1,\ldots,N$
f_d	destination node for flow f	$1,\ldots,N$
$E_I(n)$	set of edges entering node n	$\subseteq E$
$E_O(n)$	set of edges leaving node n	$\subseteq E$
$E(n)$	set of all edges incident to node n	$\subseteq E$
λ_f	traffic demand for flow f	>0
c_e	capacity of edge e	>0
S	set of all state variables	-
S_f	set of state variables needed for flow f	$\subseteq S$
H_f	set of sequence of state variable copies needed for flow f where $h \in H_f$	-
C_s	max number of copies of state variable s	≥ 1
c	c-th copy of state variable s	$1,\ldots,C_s$
h_s	copy of state variable s in sequence $h \in H_f$	$1,\ldots,C_s$
$\hat{\lambda}_s$	synch. traffic between any copy of state s	>0

Table 5.2: Output Variables

Variable	Description	Range
R_{fhe}	1 if flow f along sequence of copies h traverses edge e	Binary
$\hat{R}_{snme}$	1 if synchronization traffic from node n to node m containing copies of state variable s traverses edge e	Binary
P_{scn}	1 if copy c of state s is stored in node n	Binary
P_{fsce}	f flow f on edge e has passed copy c of state s	Binary
X_{fh}	1 if flow f is assigned $h \in H_f$	Binary
U_{sn}	1 if at least one copy of state variable s is on node n	Binary
Y_{snme}	1 if $\hat{R}_{snme} > 0$	Binary

function used by SNAP framework in [11], but without the second term since SNAP does not include any synchronization traffic.

As an alternative, the objective function can be modified to minimize the maximum congestion on a link, obtained by summing data and synchronization traffic, as

follows:

$$\min \max_{e \in E} \Big(\sum_{f \in \mathscr{F}} \sum_{h \in H_f} R_{fhe} \lambda_f + \sum_{s \in S} \sum_{n \in V} \sum_{\substack{m \in V \\ n \neq m}} \hat{R}_{snme} \hat{\lambda}_s \Big) \tag{5.2}$$

5.3.1 Constraints in the optimization problem

We discuss all the constraints considered in the ILP model. In some cases, we will get products of binary variables, but the corresponding constraint can be easily linearized with standard techniques.

Data routing constraints

The constraints (5.4)-(5.7) are similar to the constraints for the classic multi-commodity flow problem. However, our modification consists of assigning a commodity for each sequence $h \in H_f$ of state variable copies directly at the source of the flow f, in order to support the sequence of states required by each flow

We introduce an auxiliary variable, which is an indicator function X_{fh} equal to 1 if sequence $h \in H_f$ is assigned to flow $f \in \mathscr{F}$.

$$X_{fh} = \sum_{e \in E_O(f_s)} R_{fhe} - \sum_{e \in E_I(f_s)} R_{fhe} \tag{5.3}$$

Indeed, whenever a particular sequence h is adopted, similar to (5.4), the net outgoing data traffic from source f_s is 1. Notably, the second term considers the special case in which the flow is re-entering (and leaving) f_s in the path to reach the state and then the destination. We now force only one sequence h to be assigned to flow f. $\forall f \in \mathscr{F}$:

$$\sum_{h \in H_f} X_{fh} = 1 \tag{5.4}$$

A similar constraint is defined for flow f's destination f_d, but now the net incoming flow should be 1. $\forall f \in \mathscr{F}$:

$$\sum_{h \in H_f} \left(\sum_{e \in E_I(f_d)} R_{fhe} - \sum_{e \in E_O(f_d)} R_{fhe} \right) = 1 \tag{5.5}$$

The sum of all the data and synchronization traffic passing an edge must not exceed its capacity. $\forall e \in E$:

$$\sum_{f \in \mathscr{F}} \sum_{h \in H_f} R_{fhe} \lambda_f + \sum_{s \in S} \sum_{n \in V} \sum_{\substack{n^* \in V \\ n \neq n^*}} \hat{R}_{snn^*e} \hat{\lambda}_s \leq c_e \tag{5.6}$$

Finally, the standard flow conservation condition must be satisfied at any node. $\forall h \in H_f, \forall f \in \mathscr{F}$:

$$\sum_{e \in E_I(n)} R_{fhe} = \sum_{e \in E_O(n)} R_{fhe} \quad \forall n \in V \setminus \{f_s, f_d\} \tag{5.7}$$

Placement constraints

Each copy can only be placed at one switch. $\forall s \in S,\ \forall c \leq C_s$:

$$\sum_{n \in V} P_{scn} = 1 \tag{5.8}$$

We now constrain the routing to pass through the states; i.e. all the flows dependent on a copy must pass through the node where the copy is located (except at source f_s and destination f_d). $\forall n \in V \setminus \{f_s, f_d\}, \forall f \in \mathscr{F}, \forall h \in H_f, \forall s \in S_f$:

$$\sum_{e \in E_I(n)} R_{fhe} \geq P_{sh_sn} + X_{fh} - 1 \tag{5.9}$$

Indeed, if a particular sequence h is adopted for f, then (5.9) becomes $\sum_{e \in E_I(n)} R_{fhe} \geq P_{sh_sn}$ and in the case the node is storing copy h_s of state s, then $\sum_{e \in E_I(n)} R_{fhe} \geq 1$, which forces at least one R_{fhe} variable to be one on the incoming edges to e. Otherwise, if the sequence h is not adopted for f, then (5.9) becomes a useless bound.

We now define a variable that tracks the fact that a flow has traversed already a particular state along its path. For a flow f traversing copy h_s of state s, we define $P_{fsh_se} = 0$ for all the edges along the path before entering the node with copy h_s of s, and $P_{fsh_se} = 1$ for all the edges in the path after h_s. It is initialized to zero for all the

copy sequences h not used. $\forall f \in \mathscr{F}, \forall s \in S_f, \forall h \in H_f, \forall e \in E$:

$$P_{fsh_se} \leq R_{fhe} \tag{5.10}$$

To model the fact that P_{fsh_se} changes from 0 to 1 whenever the flow leaves a node where the state is stored, it holds: $\forall f \in \mathscr{F}, \forall s \in S_f, \forall h \in H_f, \forall e \in E, \forall n \in V \setminus \{f_s, f_d\}$:

$$P_{sh_sn}X_{fh} + \sum_{e \in E_I(n)} P_{fsh_se} = \sum_{e \in E_O(n)} P_{fsh_se} \tag{5.11}$$

Indeed, only when $P_{sh_sn}X_{fh} = 1$ (i.e., node n has copy h_s and f exploits h including it), the net flow of P_{fsh_se} entering n is 0 and the corresponding one leaving n is 1.

We now impose that the data flow reaches the destination f_d after having traversed all the states required in h, i.e. $P_{fsh_se} = 1$ for one edge entering f_d. $\forall f \in \mathscr{F}, \forall s \in S_f, \forall h \in H_f$:

$$P_{sh_sf_d}X_{fh} + \sum_{e \in E_I(f_d)} P_{fsh_se} = X_{fh} \tag{5.12}$$

So far, the constraints (5.10)-(5.12) force the flows to pass through all the required state variables, but not necessarily in sequence. We model here the correct sequence of traversed states, if the flow f has to cross $h_s \in H_f$ of s, followed by copy $h_{s'} \in H_f$ of s'. $\forall f \in \mathscr{F}, \forall s, s' \in S_f, \forall h \in H_f, \forall n \in V$

$$P_{sh_sn} + \sum_{e \in E_I(n)} P_{fsh_se} \geq P_{s'h_{s'}n} + X_{fh} - 1 \tag{5.13}$$

Indeed, if either flow f has been assigned sequence h i.e. $X_{fh} = 1$, or copy $h_{s'} \in H_f$ exists at node n, or copy $h_s \in H_f$ does not exist at node n, then (5.13) becomes $\sum_{e \in E_I(n)} P_{fsh_se} \geq 1$. This forces P_{fsh_se} to be 1 before entering node n, which means that the flow must have traversed h_s before entering the node containing $h_{s'}$. This ensures that the flow traverses the correct sequence of states as dictated by h.

Constraint (5.14) ensures that if flow has traversed state variable copy h_s on edge e i.e. $P_{fs'h_{s'}e} = 1$, then it must have already crossed state variable copy h_s, which ensures $P_{fsh_se} = 1$. $\forall f \in \mathscr{F}, \forall s, s' \in S_f, \forall h \in H_f, e \in E$:

$$P_{fsh_se} \geq P_{fs'h_{s'}e} \tag{5.14}$$

State Synchronization

This traffic is due to the synchronization between any pair of copies of the same state. Thanks to the routing variable $\hat{R}_{snme}$, we can model the traffic between any pair of nodes n and m containing copies of the state variable s and consider its contribution in the total traffic, as in (5.1) and (5.2), and in the constraint (5.6) regarding the edge capacity.

Since we assume that multiple copies of the state variable can be hosted on the same node n; hence to track that there is at least one copy at node n, we define the variable U_{sn} in (5.15). $\forall c \in C_s$, $\forall s \in S$, $\forall n \in V$:

$$U_{sn} \geq P_{scn} \tag{5.15}$$

For the synchronization traffic from node n to node m, the routing variable $\hat{R}_{snme}$ is treated as a commodity from node n such that $U_{sn} = 1$ to node m such that $U_{sn} = 1$. We constrain the routing to ensure the standard flow conservation equation at the intermediate node.

We define a new intermediate variable Y_{snme} which is set to 1 iff $\hat{R}_{snme} > 0$. This is ensured using the big-M method [103] as in (5.16) where M is sufficiently larger than $\hat{R}_{snme}$. $\forall s \in S$, $\forall n \in V$, $\forall m \neq n \in V$, $\forall e \in E$

$$0 \leq -\hat{R}_{snme} + MY_{snme} \leq M - 1 \tag{5.16}$$

Assume $\hat{R}_{snme} = 1 \, \forall e \in E_O(n)$, then $Y_{smne} = 1$ from (5.16). In this case, for the condition $M \geq \hat{R}_{snme}$ to be true, M must be equal to or greater than the maximum degree of G as in (5.17), where the maximum degree $\Delta(G)$ is the maximum of the number of edges incident to each node in G as shown in (5.18).

$$M \geq \Delta_O(G) \tag{5.17}$$

$$\Delta_O(G) = \max_{n \in V} |E(n)| \tag{5.18}$$

We require the egress synchronization flow from a state copy containing node to use only one outgoing edge. This can be done by exploiting Y_{snme} as in (5.19).

$\forall s \in S$, $\forall n \in V$, $\forall m \neq n \in V$:

$$\sum_{e \in E_{O(n)}} Y_{snme} \leq 1 \tag{5.19}$$

Constraints (5.20)-(5.23) are for the multi-commodity flow problem for the synchronization flows. Specifically, constraints (5.20) and (5.21) are for the originating flow from the source node m and the sink flow in the destination node m containing the state copies respectively $\forall s \in S$, $\forall n \in V$, $\forall m \neq n \in V$:

$$\sum_{e \in E_{O(n)}} Y_{snme} \geq U_{sn} \tag{5.20}$$

$$\sum_{e \in E_{I(n)}} Y_{snme} \geq U_{sm} \tag{5.21}$$

Instead, constraints (5.22)-(5.23) are for the flow conservation at intermediate nodes $\forall s \in S$, $\forall n \in V$, $\forall m \neq n \in V$:

$$\sum_{e \in E_{O(n)}} Y_{snme} \leq \sum_{e \in E_{I(n)}} Y_{snme} + U_{sn} \leq 1 \tag{5.22}$$

$$\sum_{e \in E_{I(n)}} Y_{snme} \leq \sum_{e \in E_{O(n)}} Y_{snme} + U_{sm} \leq 1 \tag{5.23}$$

5.3.2 Computational complexity

The computational complexity of the ILP model is derived in Appendix A. It is shown here below, given if all flows $f \in \mathscr{F}$ require to traverse all state variables $s \in S$, where each $s \in S$ has C number of copies:

$$O(256^{\max(N^2 C^{|S|}, |S|N^4)} \cdot \max(N|S|C^{|S|}, |S|N^4)) \tag{5.24}$$

In case there is only one state variable required by all the flows, (5.24) simplifies below:

$$O(256^{N^4} N^4) \tag{5.25}$$

5.4 Approximation algorithm for single state replication

We address specifically the problem of state replication for a single state variable. To address the limited scalability of the ILP solver, we propose PLACEMULTICOPIES algorithm which is computationally scalable and will be shown in Sec. 5.4.1 to approximate well the optimal solution obtained by the ILP solver for small problem instances.

The pseudocode of PLACEMULTICOPIES is given in Algorithm 1. It takes as input the network graph G, the state variable s and the maximum number of copies C_s of s and the set of flows $\mathscr{F}$ affecting s. As an output, the algorithm returns: the routing variables of the data flows R_{fhe} and of the state synchronization flows $\hat{R}_{smne}$ and the copies placement variables P_{scn}. The algorithm works through 3 phases:

- *Phase 1.* The network graph G is partitioned into C_s clusters, in order to minimize the maximum distance among the elements within a cluster. This allows to distribute the copies across the whole network in a balanced way, exploiting the spatial diversity offered by each cluster.
- *Phase 2.* In each cluster, a copy is placed in the *most*"*central*"*node* node with the highest betweenness centrality, in order to minimize the data traffic for each flow.
- *Phase 3.* The position of each copy is perturbed at random using a local search to improve the solution with respect to one obtained in the previous two phases.

The pseudocode of PLACEMULTICOPIES algorithm is given in Algorithm 1 with all the mentioned phases. After having initialized the routing and the copy placement variables (lines 2-4), Phase 1 is executed in line 5 by calling COMPUTEPARTITIONS. This method solves the *k-means clustering problem* [104] with $k = C_s$ using the *Lloyd's algorithm* [105] in which the node with the highest betweenness centrality is chosen as center of the partition.

As part of Phase 2 (lines 6-8), within each subgraph G_c the node n' with the highest betweenness centrality is assigned a state variable copy. As a reminder,

Algorithm 1 Placement and routing for single state placement and multiple copies

1: **procedure** $[\{R_{fhe}\},\{\hat{R}_{smne}\},\{P_{scn}\}]$ = PLACEMULTICOPIES(G, s, C_s, $\mathscr{F}$)
2: $R_{fhe} = 0, \forall f \in \mathscr{F}, h \in H_f, \forall e \in E$ ▷ Init routing
3: $\hat{R}_{smne} = 0, \forall c, g \neq c \leq C_s, \forall e \in E$ ▷ Init state sync
4: $P_{scn} = 0, \forall c \leq C_s, \forall n \in V$ ▷ Init state s location
5: $\{G_c\} \leftarrow$ COMPUTEPARTITIONS($G, C_s,$) ▷ **Phase 1:** Graph partitions $\{G_c\}$
6: **for** $c \leq C_s$ **do** ▷ **Phase 2:** Copy placement
7: $n' \leftarrow$ NODEWITHHIGHESTBC(G_c) ▷ Find best candidate in partition G_c
8: $P_{scn'} = 1$ ▷ Store the state copy location
9: $T_{\min} = \infty$ ▷ Init minimum traffic
10: **for** I iteration **do** ▷ **Phase 3:** Local search
11: $[T', \{R'_{fhe}\}, \{\hat{R}'_{smne}\}] \leftarrow$ ROUTEFLOWS($\mathscr{F}, \{P_{scn}\}$) ▷ Route flows through the copies
12: **if** $T' < T_{\min}$ **then** ▷ Check if the traffic is smaller
13: $T_{\min} = T'$ ▷ Store current best solution
14: $R_{fhe} = R'_{fhe}$ $\hat{R}_{smne} = \hat{R}'_{smne}$, $P'_{scn} = P_{scn}, \forall f \in \mathscr{F}, \forall h \in H_f, \forall c, g \neq c \leq C_s, \forall e \in E, \forall n \in V$
15: $\{P'_{scn}\} \leftarrow$ PERTURBCOPYLOCATION($\{P_{scn}\}$) ▷ Change existing location of state copies
16: **return** $[\{R_{fhe}\},\{\hat{R}_{smne}\},\{P_{scn}\}]$

17: **procedure** $[T_{\text{CURRENT}}, R'_{fce}, \hat{R}'_{smne}]$ = ROUTEFLOWS($\mathscr{F}, P_{scn}$)
18: $T_{\text{current}} = 0$ ▷ Init total traffic
19: **for** $f \in \mathscr{F}$ **do** ▷ For each flow
20: minDist $= \infty$ ▷ Init minimum distance
21: $c_b \leftarrow$ null ▷ Init best copy for current flow
22: $\mathscr{P}_{best} \leftarrow$ null ▷ Path with minimum length for $f_s \rightarrow n_c \rightarrow f_d$
23: **for** $c \in C_s$ **do** ▷ For all state copies
24: $\mathscr{P} =$ SHORTESTPATH(f_s, n_c) $\cup$ SHORTESTPATH(n_c, f_d)
25: **if** $\mathscr{P}$.length $<$ minDist **then**
26: minDist $=$ $\mathscr{P}$.length ▷ Update minimum distance
27: $\mathscr{P}_{best} \leftarrow \mathscr{P}$ ▷ Store path with minimum length
28: $c_b \leftarrow c$ ▷ Store best copy for this flow
29: **for** $e \in \mathscr{P}_{best}$ **do** ▷ For each edge in the minimum length path
30: $R'_{fc_be} = R'_{fc_be} + \lambda_f$ ▷ Store the routing
31: $T_{\text{current}} = T_{\text{current}} + \lambda_f$ ▷ Store the traffic value
32: **for** $c \in C_s$ **do** ▷ For each c$^{\text{th}}$ copy of state variable s
33: **for** $g \neq c \in C_s$ **do** ▷ For each g$^{\text{th}}$ copy of state variable s
34: $\mathscr{P}_{cg} \leftarrow$ SHORTESTPATH(n_c, n_g) ▷ Shortest path from $n_c \rightarrow n_g$
35: **for** $e \in \mathscr{P}_{cg}$ **do** ▷ For each edge in the path $n_c \rightarrow n_g$
36: $\hat{R}_{smne} = \hat{R}_{smne} + \alpha$ ▷ Store the state sync flow
37: $T_{\text{current}} = T_{\text{current}} + \alpha$ ▷ Update total traffic
38: **return** $[T_{\text{current}}, R'_{fce}, \hat{R}'_{smne}]$

betweenness centrality of a node v is proportional to the number of shortest paths crossing it.

Lines 10 to 15 refer to a local search procedure with I iterations. Within each iteration, ROUTEFLOWS is used to route flows through the location of the copies identified in Phase 2, following two sub-paths: one from the flow source node to the closest copy and one from this copy to the destination node. The procedure works on the set of flows $\mathscr{F}$ and the location of state variables P_{scn} and returns the routing variables for data flows R'_{fce} and for state synchronization $\hat{R}'_{smne}$, and the corresponding total traffic T' in the network. Lines 19 to 31 route the data flows from their source f_s to the destination f_d while traversing the copy c_b which has

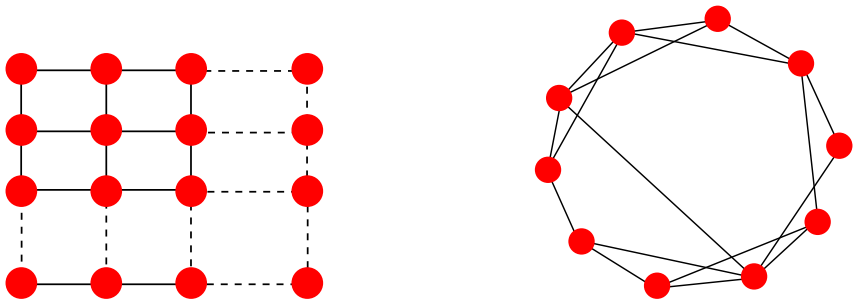
the minimum path length among all other copies. For each flow, in lines 21 and 29, the copy c_b and the path $\mathscr{P}_{best}$ traversing it are initialized. Then for each copy (in lines 23-28), first, the shortest path $f_s \rightarrow n_c \rightarrow f_d$ is computed. n_c is the vertex for which $P_{scn} = 1$. If the path length $\mathscr{P}.length$ is less than the previous minimum *minDist* in line 25, then the current path $\mathscr{P}$ is stored as the best path $\mathscr{P}_{best}$ and the current copy c as the best copy c_b. In lines 29-31, for each edge in $\mathscr{P}_{best}$, the routing as well as the traffic value is updated. Lines 32 to 37 generate flows from each state copy c to all the other state copies g for state synchronization using the shortest path. This includes the synchronization flows $\hat{R}_{scge}$ being updated in line 36 for each edge in the path $\mathscr{P}_{cg}$ before updating the total traffic in line 37. If T' is less than the previous minimum, then the minimum traffic value and all the decision variables are updated (lines 13-14). In Phase 3 (line 15), a local search procedure perturbs the existing state copy locations. This proceeds by randomly selecting one node where a copy is located and moving it to one of its neighbor nodes. This new solution is then compared with the current one (line 12) after having evaluated the corresponding routing and total traffic.

5.4.1 Performance comparison

We evaluate the performance of PLACEMULTICOPIES presented in Sec. 5.4. In the case of small instances of the problem, we run an ILP solver, coded using IBM CPLEX Optimizer [106], implementing the model in Sec. 5.3. We computed the *approximation ratio*, i.e. the ratio between the total traffic obtained by PLACEMULTICOPIES and the one obtained by the ILP solver. We consider two standard topologies for the network graph: unwrapped Manhattan and Watts-Strogatz:

- Manhattan unwrapped: This is a $N \times N$ sized topology in a grid as shown in Fig. 5.2a.
- Watts-Strogatz [107]: adds a few long-range links to regular graph topologies that reduce the shortest path between two nodes and emulate a small-world model with an example shown in Fig 5.2b. It is generated by taking a ring of n nodes, where each node is connected to k-nearest neighbors. A node is chosen at random and the edge connected to its nearest clockwise neighbor is disconnected with probability p and connected to another node chosen uniformly at random over the entire ring. This was chosen as a test topology

since it produces connected graphs having a fixed average degree. Here, $p = 0.1$ and average degree of 8 was chosen.



(a) Manhattan unwrapped (b) Watts-Strogatz

Fig. 5.2: Topologies used for performance evaluation of state replication

The local search in PLACEMULTICOPIES runs with $I = 1000$ iterations. We utilize random traffic matrices with the number of flows equal to the number of vertices in the graph, i.e. $|\mathscr{F}| = N$ and with unity demands ($\lambda_f = 1$). The source-destination pairs for the flows were generated according to two models. In the case of *uniform traffic*, all the source nodes were associated to a random permutation of nodes as destination; thus each node is source and destination of exactly one flow. In the case of *clustered uniform traffic*, we partitioned the graph in half and generated a random permutation between the nodes of the same partition; thus all the flow are defined within the same partition. All the results were obtained with 1000 different runs to get very small 95% confidence intervals (in all cases within 4.2% accuracy).

Synchronization rate and number of copies

In Fig. 5.3 we evaluate the effect of varying the number of copies for state s and of the synchronization rate $\hat{\lambda}_s$, through the optimal ILP solver. We consider a 4×4 Manhattan graph and set $C_s = 7$. As expected, when increasing $\hat{\lambda}_s$, the number of copies reduces, since the higher costs of synchronization decreases the beneficial effect of multiple copies on the data traffic. Instead the synchronization traffic is almost constant, since, for smaller number of copies, their relative distances grows, to "cover" a larger area of the network. As a term of comparison, we report the total traffic for one single copy allowed in the network, as considered in the original SNAP framework. In this case, the total traffic is constant with respect to $\hat{\lambda}_s$, since there is no synchronization traffic. By comparing the results obtained with multiple

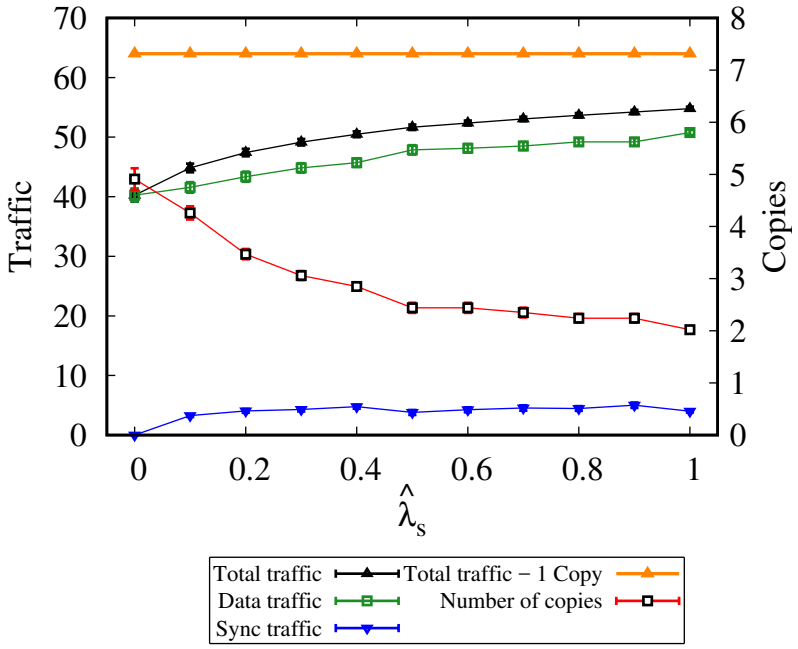


Fig. 5.3: Optimal traffic and number of copies in a 4×4 Manhattan graph for uniform traffic, using the ILP solver

copies, the reduction in the total traffic in the network can reach 33% thanks to state replication.

Fig. 5.4 is used to depict the effect of a very large $\hat{\lambda}_s$, when the synchronization traffic has a larger weight than the data traffic in the objective function. At the value of $\hat{\lambda}_s = 6.1$, the number of copies used in the network is equal to 1. This shows that having a very high synchronization traffic is detrimental, and there is a threshold to which the state replication is beneficial.

Comparison with ILP

Figs. 5.5-5.6 show the approximation ratio for different number of nodes, copies and different values of $\hat{\lambda}_s$, under uniform traffic. The two graphs refer to Manhattan and Watts-Strogatz topologies. The approximation ratio in all cases is always ≤ 1.15, thus PLACEMULTICOPIES approximates well the ILP solution. For larger topologies, we cannot provide the approximation ratio as the ILP optimization is not computationally feasible.

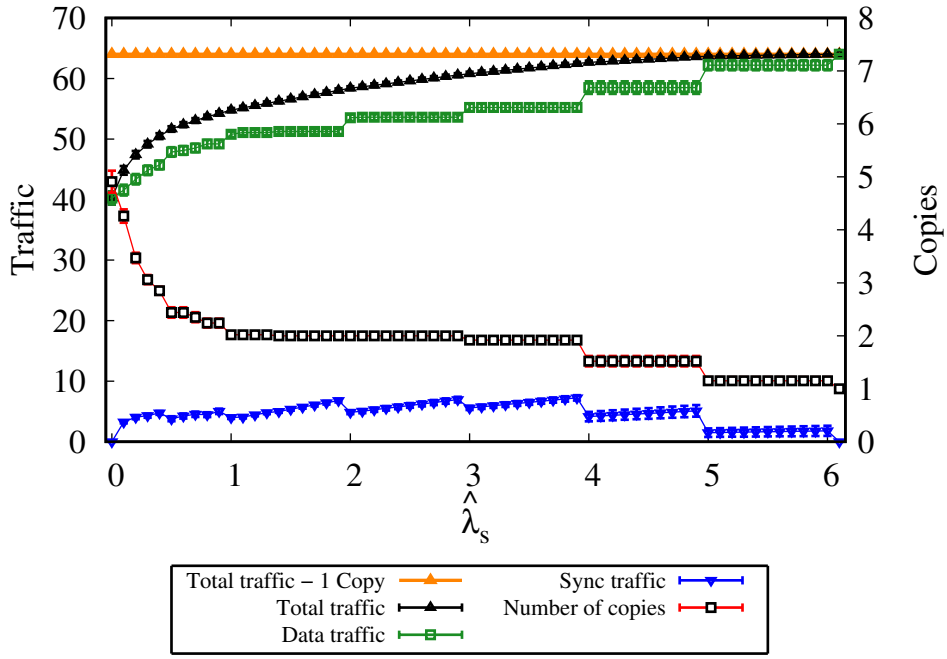


Fig. 5.4: An extended version of Fig. 5.3 where $\hat{\lambda}_s$ ranges from 0.0 to 6.1

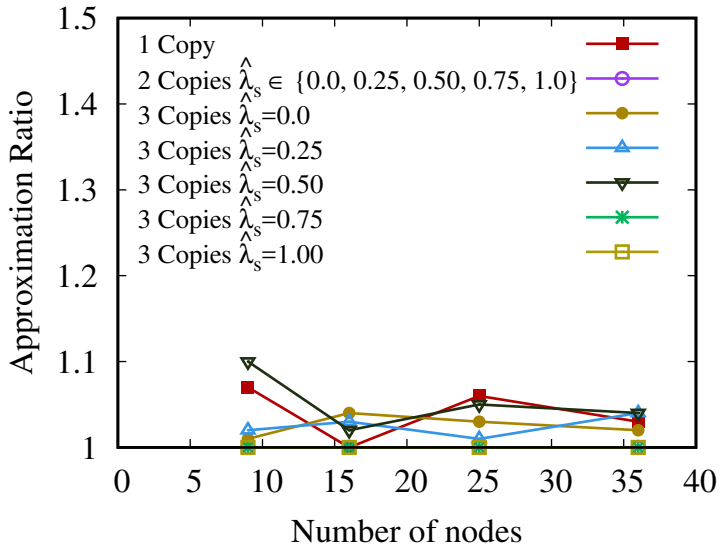


Fig. 5.5: Approximation ratio of PLACEMULTICOPIES in a Manhattan graph under uniform traffic.

Number of copies in large topologies

We run PLACEMULTICOPIES algorithm on large topologies. Figs. 5.7-5.8 show the total traffic normalized to the number of flows for variable-size Manhattan and

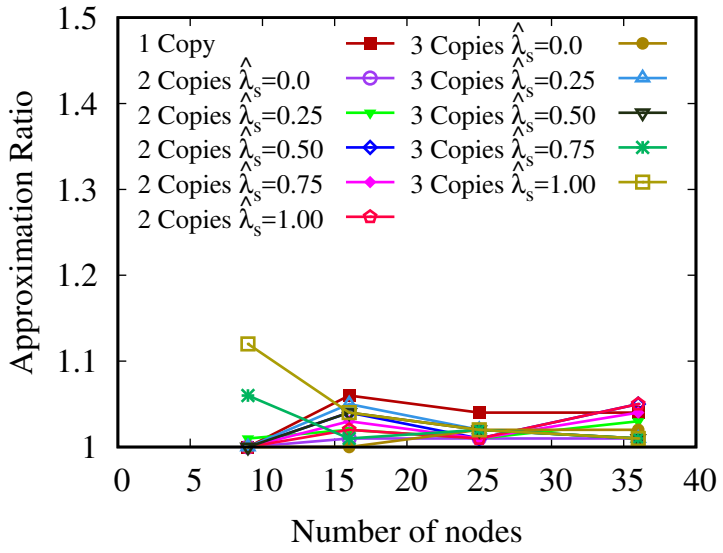


Fig. 5.6: Approximation ratio of PLACEMULTICOPIES in Watts-Strogatz graph under uniform traffic.

Watts-Strogatz topologies, under clustered uniform traffic. We set $\hat{\lambda}_s = 0.5$. For comparison, we also report the result of the traffic obtained by routing each flow from its source to its destination along the shortest path, obliviously of the placement of the state copies; this provides a lower bound on the total traffic in the network obtained for the optimal solution of the ILP problem (which cannot be computed in this case).

As expected, the highest amount of traffic is given by the single copy case, i.e. SNAP, because of the longer path to reach the state location experienced by all the flows. Now adding one copy provides a beneficial effect, since the spatial diversity of 2 copies can be exploited to route the flows and minimize the total traffic. The gain is generally around 30% for Manhattan graph and grows up to 20% in Watts-Strogatz graph. If increasing again the number of copies from 2 to 3, then the gain is very limited (around 5%), since the higher spatial diversity is compensated by a higher synchronization traffic.

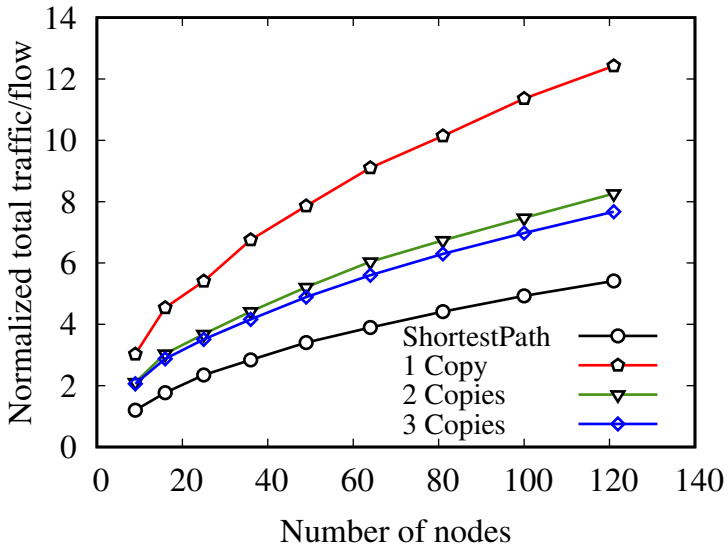


Fig. 5.7: Performance of PLACEMULTICOPIES in Manhattan graph under clustered uniform traffic

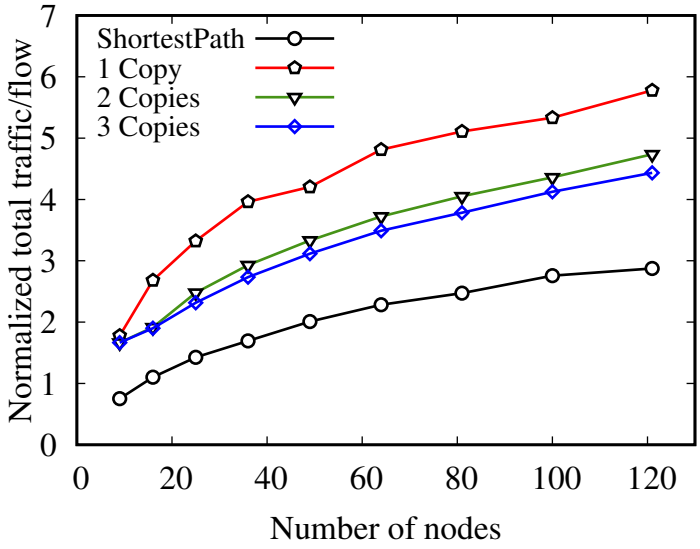


Fig. 5.8: Performance of PLACEMULTICOPIES in Watts-Strogatz graph under clustered uniform traffic

5.5 Asymptotic analysis for unwrapped Manhattan topology

In this section, we discuss the methodology to compute the required number of copies, given the size of a graph using asymptotic analysis. In our case, we present this method for the unwrapped Manhattan topology. We model this problem, similar to how traffic is routed in an unwrapped Manhattan topology through C copies, taking as example the PLACEMULTICOPIES algorithm.

5.5.1 Methodology

We assume that the number of copies C is a perfect square, i.e. $\sqrt{C} \in \mathbb{N}$. Consider a unit square as shown in Fig. 5.9, representing the boundary of the unwrapped Manhattan topology containing N nodes with $N \to \infty$. The unit square is divided into individual C squares, with each side having a dimension of $1/\sqrt{C}$ and each having a center point P_c^{ctr}, where $c = 1 \ldots C$. P_c^{ctr} denotes the location of a state copy in the network, where $P_c^{ctr}(x)$ and $P_c^{ctr}(y)$ denote the x and y coordinates respectively . The minimum number of copies can be obtained for the case which minimizes the total traffic in the topology.

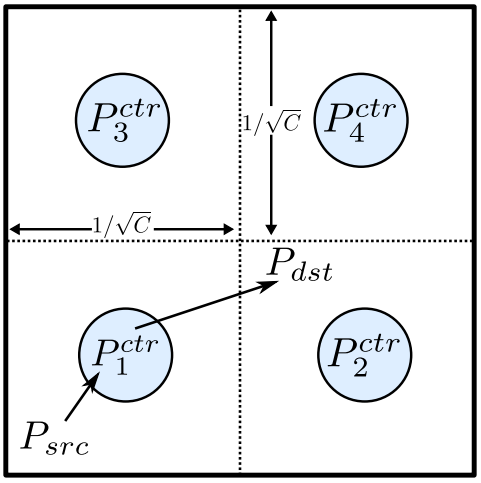


Fig. 5.9: Unwrapped Manhattan topology represented as a unit square

The total traffic is composed of the data traffic and the synchronization traffic. Given a traffic demand in this scenario, we assume that the traffic is routed in

a straight line between two points in the square instead of in a step-wise stair-like manner. This is due to the fact that we assumed that the graph size is very large ($N \to \infty$). To compute the data traffic, we utilize the Monte Carlo method to compute the average distance traveled between two nodes. Two points with random coordinates are generated in the unit square which are P_{src} and P_{dst} for source and destination nodes respectively as shown in Fig. 5.9. The distance from P_{src} to the closest P_c^{ctr} and from P_c^{ctr} to P_{dst} is added to compute the distance traveled between the source and destination nodes. These distances are generated for 10^7 iterations and are averaged as $\hat{d}_{data}$ for each number of copies. It is shown in Fig. 5.10. This average distance asymptotically approaches 0.5412 [108].

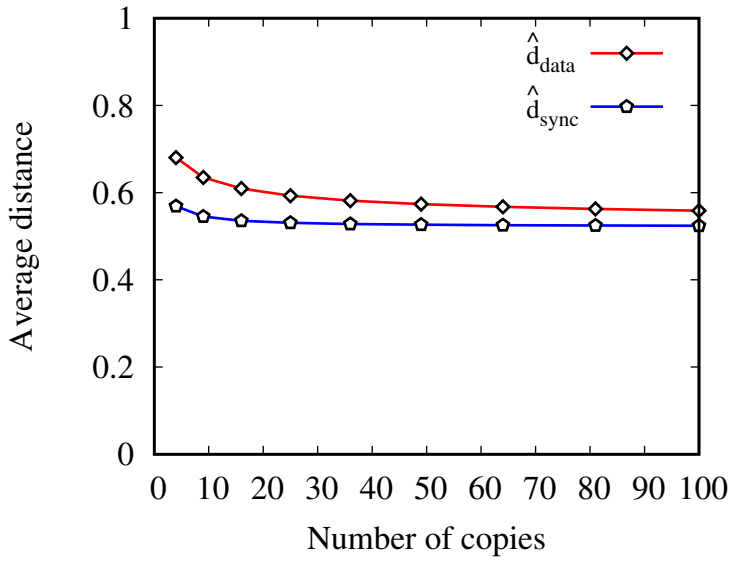


Fig. 5.10: Average distance vs. C for a unit square

The data traffic can be computed by (5.26):

$$T_{data} = \lambda_f \times N \times \hat{d}_{data} \tag{5.26}$$

where we assume that the total number of flows is $|\mathscr{F}| = N$ as in previous results.

Similarly for the synchronization traffic, the average distance is computed for each number of copies c using (5.27). Here, no random points are generated since the copies have a fixed position in the unit square.

$$\hat{d}_{sync}(C) = \frac{\sum_{c=1}^{C}\sum_{\substack{c'=1 \\ c'\neq c}}^{C} \sqrt{(P_c^{ctr}(x) - P_{c'}^{ctr}(x))^2 + (P_c^{ctr}(y) - P_{c'}^{ctr}(y))^2}}{C(C-1)} \tag{5.27}$$

Using (5.27), the average distance between any two copies $\hat{d}_{sync}$ asymptotically approaches 0.5221 as shown in Fig. 5.10. The synchronization traffic can be computed by (5.28).

$$T_{sync} = \hat{\lambda}_s \times \hat{d}_{sync} \times C(C-1) \tag{5.28}$$

Using (5.26) and (5.28), the total traffic for a Manhattan unwrapped topology of size N is given in (5.29). The right hand side is scaled by $\sqrt{N}$ since the distances $\hat{d}_{data}$ and $\hat{d}_{sync}$ were computed for a unit square, whereas T_{TOT} is for a square with a side of size $\sqrt{N}$.

$$T_{TOT} = \sqrt{N}(\underbrace{\lambda_f N \hat{d}_{data}}_{\text{data traffic}} + \underbrace{\hat{\lambda}_s \hat{d}_{sync} C(C-1)}_{\text{synchronization traffic}}) \tag{5.29}$$

Note that in (5.29), the rate of synchronization traffic does not depend on the amount of data traffic traversing state copy based on our model.

For different values of N and $\hat{\lambda}_s/\lambda_f$ where $\lambda_f = 1$, the number of copies C for which the minimum T_{TOT} is obtained is given in Fig. 5.11.

Note that for higher values of N, more copies are required to cover the network. Thus, as $N \to \infty$, then $C \to \infty$ as well. For higher values of $\hat{\lambda}_s/\lambda_f$, the number of copies decreases since more synchronization traffic is required. Fig. 5.11 is shown as a log-log plot where each line fitted to the points can be represented as shown in (5.30) $\forall \hat{\lambda}_s/\lambda_f$.

$$\log_{10} C = m \log_{10} N + k \tag{5.30}$$

Taking an anti-log on both sides of (5.30) gives (5.31):

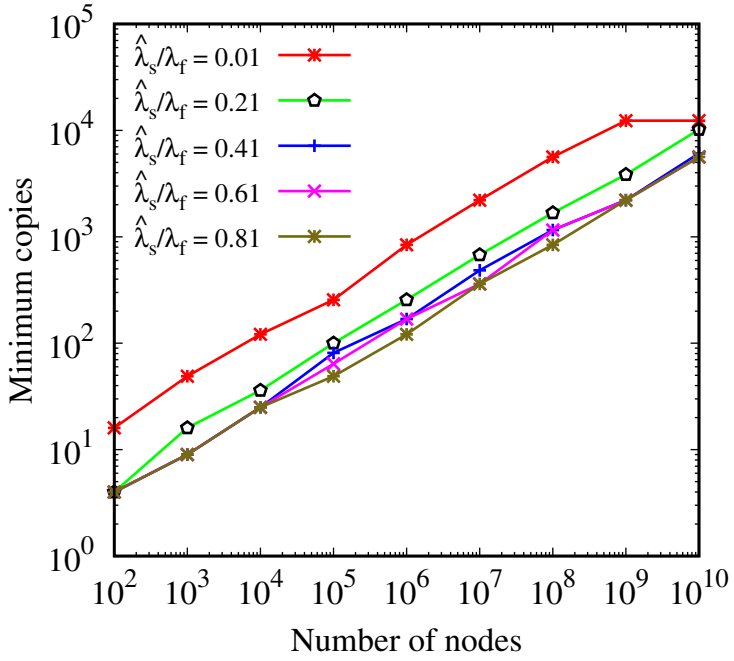


Fig. 5.11: Minimum copies required vs. N for different $\hat{\lambda}_s/\lambda_f$

$$C = 10^k N^m \tag{5.31}$$

This indicates that the slope of a straight line in Fig. 5.11 is the exponent m and the y-intercept is the constant k. Using this methodology, the relation of C with N can be computed using (5.31) by using the values of 10^k and m, $\forall \hat{\lambda}_s/\lambda_f$, given in Fig. 5.12.

To further generalize (5.12), different $\hat{\lambda}_s/\lambda_f$, N and the corresponding 10^k and m from Fig. 5.12 are substituted in (5.31). This results in a three dimensional plot along with the curve-fitted plane as shown in Fig. 5.13.

The curve fitted plane has the form as shown in (5.32).

$$\log_{10} C = x + y \log_{10} N + z \log_{10} \left(\frac{\hat{\lambda}_s}{\lambda_f} \right) \tag{5.32}$$

where,

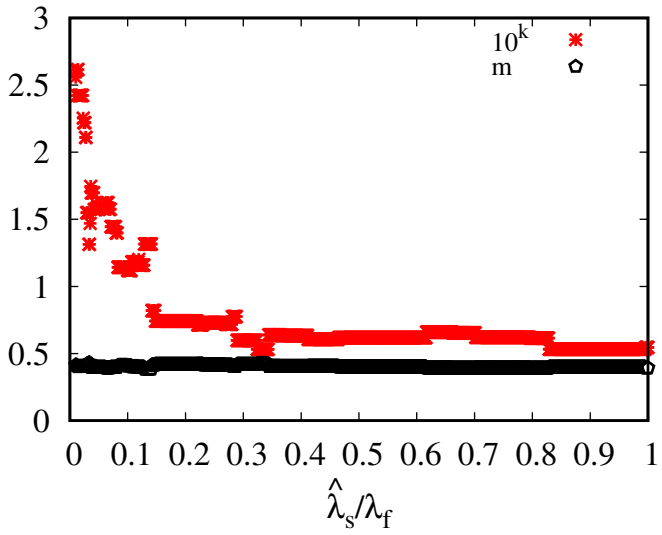


Fig. 5.12: 10^k and m vs. $\hat{\lambda}_s/\lambda_f$

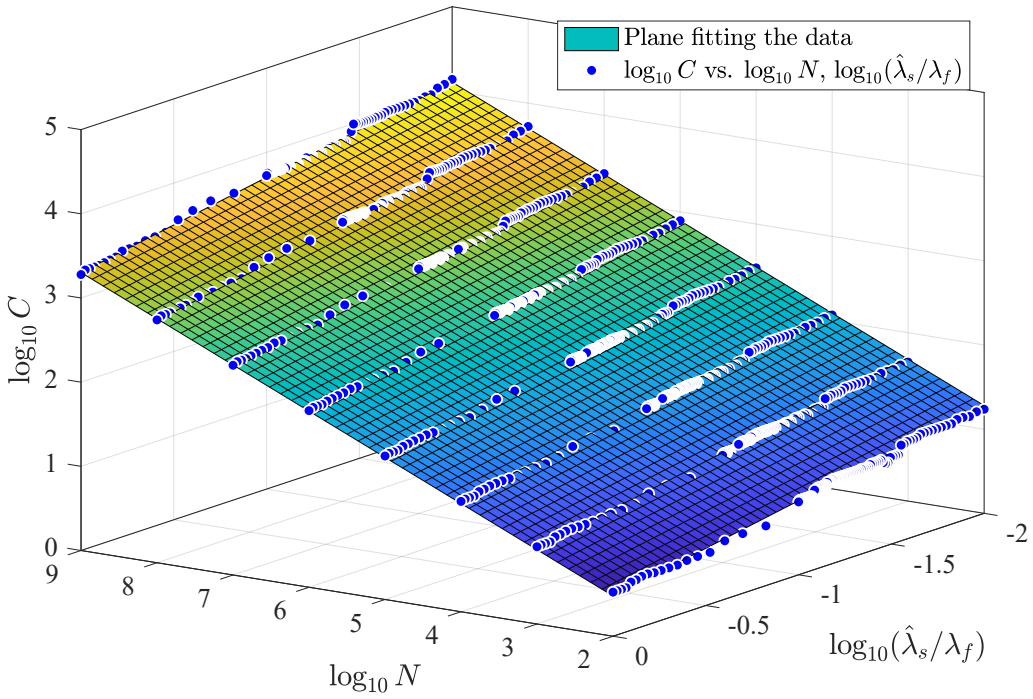


Fig. 5.13: $\log_{10}(C)$ vs. $\log_{10}(\hat{\lambda}_s/\lambda_f)$ and $\log_{10} N$

$$C = 10^x N^y \left(\frac{\hat{\lambda}_s}{\lambda_f}\right)^z \tag{5.33}$$

Using curve fitting, the coefficients of (5.33) were computed and shown in (5.34).

$$C = 0.47N^{0.40}\left(\frac{\hat{\lambda}_s}{\lambda_f}\right)^{-0.40} \tag{5.34}$$

For verification, values of $\hat{\lambda}_s/\lambda_f$ are substituted in (5.34) and the coefficient 10^k is computed and compared with the one in Fig. 5.12 as shown in Fig 5.14. The curve is fitting the points, thus proving that (5.34) is reliable.

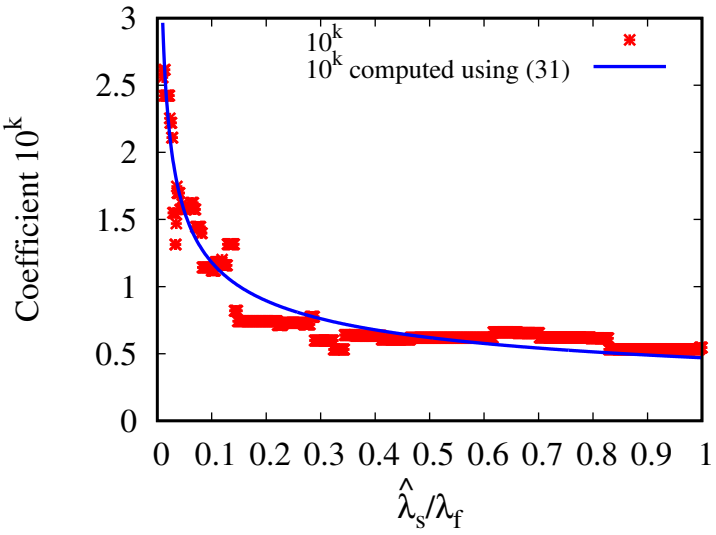


Fig. 5.14: Comparison of original 10^k vs. 10^k derived from (5.34)

Since the number of copies must be a positive integer number, hence we modify (5.34) in (5.35).

$$C_{comp} = \left\lceil 0.47N^{0.40}\left(\frac{\hat{\lambda}_s}{\lambda_f}\right)^{-0.40} \right\rceil \tag{5.35}$$

(5.35) can be used to compute the number of copies C_{comp} required for an unwrapped Manhattan topology of size N, given N number of flows, with each flow having a traffic demand of $\lambda_f = 1$. This methodology can be further extended for other topologies, using their particular average distances between two points in the topology.

5.5.2 Results

We present here the results of minimum number of copies required for the unwrapped Manhattan topology using the optimization model from Sec. 5.3 and the heuristic PLACEMULTICOPIES as C_{opt} and C_{heur} respectively, vs. the minimum number of copies C_{comp} obtained using (5.35). *Uniform traffic* was used as described in Sec. 5.4.1 with all results obtained with 1000 different runs to get very small 95% confidence intervals. Figs. 5.15-5.16 show the difference of the computed values C_{comp} from (5.35) for different sizes of the unwrapped Manhattan topology and different values of $\hat{\lambda}_s/\lambda_f$, with respect to C_{opt} and C_{heur}, using the optimization model and the heuristic respectively. In all the cases, the minimum number of copies from (5.35) is the same or one copy more than the result obtained from the optimization or heuristic. This shows that (5.35) can approximate the number of copies required for unwrapped Manhattan topology.

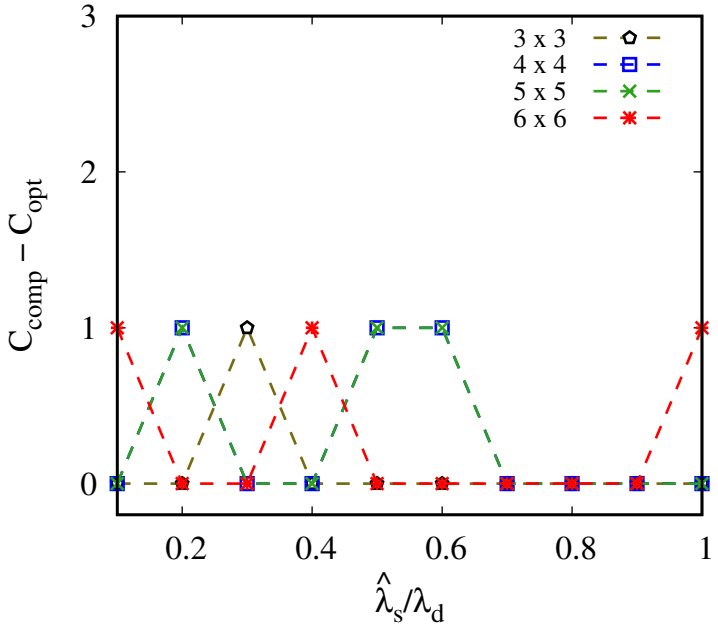


Fig. 5.15: Difference of minimum number of copies computed using (5.35) (C_{comp}) vs. optimization (C_{opt}) for unwrapped Manhattan topology

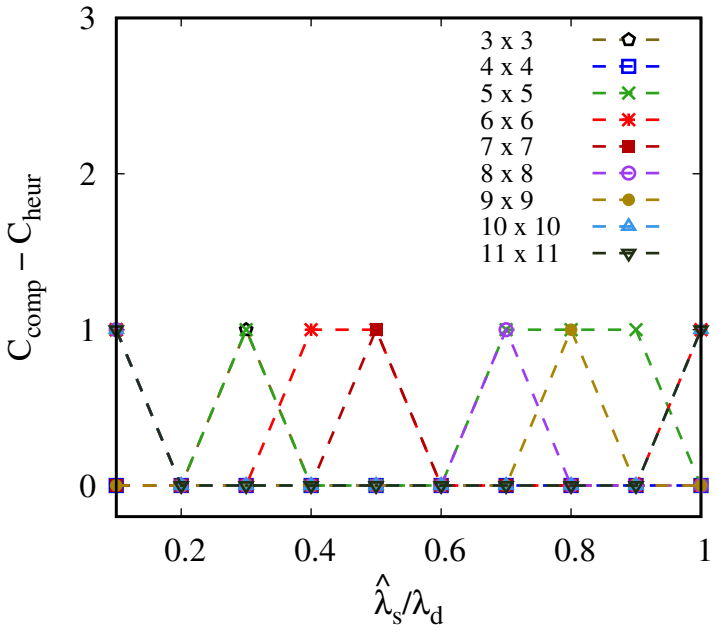


Fig. 5.16: Difference of minimum number of copies computed using (5.35) (C_{comp}) vs. heuristic (C_{heur}) for unwrapped Manhattan topology

5.6 Related work

The Virtual Network Embedding (VNE) problem finds the optimal placement of chains of VNFs while providing various optimization metrics. VNE can be closely mapped to the problem mentioned in this paper, if we consider network functions to be states and chains to be dependency graphs as computed by SNAP. There exist multiple ILP formulations and heuristics for VNE (an extensive survey is available in [109]), some of which are similar to the one proposed in our work. However, to our best knowledge, none of them consider the possibility of having replicated and coordinated virtual functions.

As mentioned before, SNAP [11] is a novel network programming abstraction, which allows to define quite complex network applications for stateful SDN and solves the problem of how to optimally place the states across the network switches, taking into account the dependency between states and the traffic flows. By design, SNAP is limited to just one replica of each state within the network. Our work, instead, enables multiple replicas of the state, extending the single replica approach in SNAP.

The most relevant work to the state replication in stateful data planes is Swing State [110], which introduces a mechanism providing state migrations entirely in the data plane but, similarly to SNAP, assumes only one state that is on demand migrated across the network.

Regarding the implementation of stateful SDN, OpenState [111] proposed a minimal architectural extension to the OpenFlow data plane and control plane to identify the flow to which a packet belongs to and to retrieve/update the associated state. OPP [7] extended OpenState by adding additional features that allow the executions of Extended Finite State Machines (EFSM) directly in the data plane. FAST [6] proposes a switch chip implementation based on the Reconfigurable Match Tables (RMT) model that permits, even if with some limitations indicated by the same authors, to manipulate some state within its pipeline. Finally, Domino/Banzai [112] proposes both a domain specific language and a data plane architecture for designing and implementing line-rate stateful processing. To read/write states, a set of specialized stateful processing instructions are executed within the switch pipeline.

NetPaxos [113] provides application layer acceleration for Paxos protocol by offloading parts of it to the network. Differently from NetPaxos, our work focuses on providing a mean for line-rate state replication directly in the data plane.

5.7 Summary

We considered the state replication problem in SNAP and proposed a MILP formalization of the optimal placement of the state replicas and of the corresponding traffic routing, taking into account the synchronization traffic exchanged among the state replicas. Furthermore, we designed a scalable algorithm to solve the problem and proved numerically that it can approximate well the optimal solution obtained through a MILP solver. We numerically showed the beneficial effect of state replication on reducing the overall traffic in the network. Our asymptotic analysis presented a methodology to compute the required minimum number of state replicas to cover an unwrapped Manhattan topology.

Chapter 6

Conclusion

In this thesis, we discuss novel applications and analysis of the SDN control plane interfaces and optimizing stateful data planes by utilizing replicated states. Specifically, in Part I, we target the SDN control plane; where we analyse the impact of data plane events on the inter-controller traffic; we propose a novel application utilizing time-synchronized operations in software-defined elastic optical networks to reduce lightpath disruption time by using the south-bound protocol extensions; and we utilize the SDN north-bound interface in the form of inter and intra-domain network service orchestration. In Part II, we propose using replicated states in a stateful SDN data plane which reduces the overall traffic in the network as opposed to a single-state approach.

In Chapter 2, we focus on the east-west interface between distributed SDN controllers. We begin by emphasizing the importance of the inter-controller traffic exchanged among the controllers. The inter-controller traffic is used to maintain a logically centralized view of the network by making the controllers consistent with each other. For this thesis, we utilize ONOS as the distributed SDN controller. ONOS includes distributed data stores, which are replicated among all the controllers in a cluster using the inter-controller traffic. Specifically, we investigate the contribution to the inter-controller traffic due to the network state, i.e. the topology, flows and the host store. The topology store is made consistent using an anti-entropy protocol; where we experimentally demonstrate that the inter-controller traffic due to the topology store increases linearly as a function of the data plane switches and links. Since it is based on the anti-entropy protocol which generates periodic

traffic, the inter-controller remains increased due to the addition of switches/links. We empirically derive the bandwidth equation for a 2-controller and 3-controller scenario which can be used to compute the amount of traffic between the controllers, based on the number of switches and links in the network topology. The methodology used to derive the throughput equation can be applied to any number of controllers. We then investigated the impact of the OpenFlow based flow rules on the inter-controller traffic. When ONOS detects a change in the flow table of the device, it sends a replica of the entire flow table of the device to its first slave controller, which generates a considerable amount of traffic. We reported the maximum inter-controller traffic generated due to some commercial OpenFlow switches due to the flow store updates for OpenFlow 1.0 and OpenFlow 1.3. We then assess the impact of host store on the inter-controller traffic, where the host store is made consistent among the controllers using a strongly-consistent RAFT consensus protocol. This traffic is event-driven and exchanged only when there is an addition/removal of hosts. We provide minimum and maximum bounds on the amount of inter-controller traffic generated due to the addition of a host per RAFT partition, independently from the number of controllers in a cluster. Based on these results, a network designer can carefully design and dimension the network infrastructure which carries the inter-controller traffic.

In Chapter 3, we focus on the south-bound interface of the SDN controller while targeting software-defined elastic optical networks. We propose utilizing the south-bound extensions of OpenFlow and NETCONF protocols which consist of time-synchronized operations to perform lightpath reassignment while reducing the disruption time. We begin by highlighting an end-of-line scenario in elastic optical networks, where the rerouting of lightpaths is inevitable, in order to cater a new lightpath request. The lightpath rerouting can be done either in an Asynchronous (ASY) approach or using Time-Synchronized Operations (TSO), where we advocate that TSO reduces the lightpath disruption time. TSO involves simultaneous operations on optical nodes using timestamps within the south-bound protocols. This is possible in OpenFlow which natively supports bundling multiple operations to be executed at a given time. However NETCONF, even though having the ability to execute operations at a particular time, needs an intelligent agent to execute simultaneous operations. We analytically depict the performance of TSO vs ASY, and show that TSO reduces lightpath disruption by a minimum of 75% as compared to the ASY approach. The analytical performance is followed by an experimental validation

of TSO in a five-node metropolitan optical network test-bed, while assuming an end-of-line scenario. An initial state and the final state of the network is compared for both ASY and TSO approach in terms of the OSNR, power budget and spectrum tilt which very close for both approaches.

In Chapter 4, we focus on the north-bound interface of the SDN controller to provision network connectivity between VNFs as a part of a network service orchestrated by a Management and Network Orchestration (MANO) system. We present work conducted for inter and intra-domain network orchestration. For the inter-domain orchestration, we discuss the 5G UK Exchange (5GUKEx), which is a lightweight network service orchestration platform. Different operators can integrate their ETSI-based MANO system to 5GUKEx. They expose their network services, and 5GUKEx acting as a broker, performs multi-operator network service instantiation and interconnects the VNF chain in different administrative domains using its inter-domain connectivity infrastructure. 5GUKEx being a lightweight platform, only performs network service orchestration and leaves the computationally intensive resource orchestration to the local operators; in this way the operators do not need to expose their network infrastructure while preserving their confidentiality. Furthermore, we experimentally demonstrate the scalability of the 5GUKEx by emulating operator MANO systems. For the intra-domain orchestration, we extended the ETSI-compliant Open Source MANO with a Transport-API (T-API) based WIM connector. This allows OSM to orchestrate network services with VNFs running on multiple PoPs. Here, we connect the VNFs using a heterogeneous network based on packet and optical switches. The WAN Infrastructure Manager (WIM) utilizes a T-API based north-bound API to interconnect the VNFs.

In Chapter 5, we focus on the stateful data planes in the context of SDN. Stateful data planes consist of data plane switches which maintain some state of the network. We emphasize that a stateful data plane improves the reactivity for network applications, and avoids the latency overhead caused by the interaction with an SDN controller. We discuss the limitation of SNAP, a previous contribution in the case of stateful data planes, and our system of replicated states improves upon what SNAP offers. SNAP introduces a platform which decides the placement of state variables in the network, and routing flows to traverse the required states. However, SNAP allows only one unique position per state variable in the network. Our approach instead, advocates for replicas of the state variable in the network, while catering for the synchronization between them. In this way, the overall traffic in the network

is reduced and there is a better balance of traffic. Our work consists of an ILP formulation of the optimal placement of state replicas and routing of flows through the closest state copy. Furthermore, we present a simple heuristic for the placement of state copies based on a graph-partitioning method. The heuristic performs well, having an approximation ratio always ≤ 1.15 for smaller network topologies. For larger topologies, we observe significant decrease of traffic for 2 copies as compared to a single state variable copy. Moreover, we present an asymptotic analysis to derive an equation to compute the number of copies required for an unwrapped Manhattan topology, under certain conditions.

In summary, we presented innovative work using all the interfaces of an SDN controller; where we analyzed the east-west interface traffic and provided novel applications utilizing the south-bound and north-bound interface. Also, we presented the concept of replicated states in a stateful SDN data plane.

Appendix A

Stateful replication ILP model

A.1 Computational complexity

Here, we discuss the computational complexity to solve the ILP model shown in Sec. 5.3. The complexity to solve an ILP model is $O(2^{2^{v+2}}d)$ [114], which can be simplified to $O(256^v d)$; where v is the number of variables and d is the number of constraints.

Number of variables

The variables in this case will be derived from Table 5.2. For R_{fhe}, the size of set H_f is shown in (A.1), $\forall f \in \mathscr{F}$.

$$|H_f| = \prod_{s \in S_f} C_s \tag{A.1}$$

Hence, the variables and their quantity is shown in Table A.1. Similarly, the number of constraints is shown in Table A.2.

The total number of variables using Table A.1 is shown in (A.2).

$$v = (|E|+1) \sum_{f \in \mathscr{F}} \prod_{s \in S_f} C_s + N\left(|S| + \sum_{s \in S} C_s\right) + 2N^2|E||S| \tag{A.2}$$

Table A.1: Number of output variables

Variable	Quantity
R_{fhe}	$\lvert E\rvert \sum_{f\in\mathscr{F}} \prod_{s\in S_f} C_s$
$\hat{R}_{snme}$	$\lvert E\rvert\lvert S\rvert N^2$
P_{scn}	$N \sum_{s\in S} C_s$
P_{fsce}	$\lvert E\rvert \sum_{f\in\mathscr{F}} \sum_{s\in S_f} C_s$
X_{fh}	$\sum_{f\in\mathscr{F}} \prod_{s\in S_f} C_s$
U_{sn}	$\lvert S\rvert N$
Y_{snme}	$\lvert E\rvert\lvert S\rvert N^2$

Assume all flows $f \in \mathscr{F}$ require all state variables $s \in S$, and each state variable s has C copies. Furthermore, the maximum number of both edges E and flows $\mathscr{F}$ is $N(N-1)$. Using this information, v in (A.2) is upper bounded by:

$$v \leq N^2 C^{|S|} + N(|S| + |S|C) + 2N^2 \cdot N^2|S| \tag{A.3}$$

Neglecting terms, (A.3) grows as:

$$O(N^2 C^{|S|}, |S|N^4) \tag{A.4}$$

Number of constraints

The total number of constraints using Table A.2 is shown in (A.5).

$$\begin{aligned} d = 2|\mathscr{F}| + |E| + [N+1+|E|\,(N-1)] \sum_{f\in\mathscr{F}} \prod_{s\in S_f} C_s + (N+1) \sum_{s\in S} C_s \\ + (N+1) \sum_{f\in\mathscr{F}} |S_f - 1| \prod_{s\in S_f} C_s + N(N-1)|S|(7+2|E|) \end{aligned} \tag{A.5}$$

Using the same assumptions to derive (A.3), the number of constraints in (A.5) is upper bounded by:

Table A.2: Number of constraints

Constraint equation	**Quantity**
(5.3)	$\sum_{f\in\mathscr{F}}\prod_{s\in S_f} C_s$
(5.4)	$\lvert\mathscr{F}\rvert$
(5.5)	$\lvert\mathscr{F}\rvert$
(5.6)	$\lvert E\rvert$
(5.7)	$\sum_{f\in\mathscr{F}}\prod_{s\in S_f} C_s$
(5.8)	$\sum_{s\in S} C_s$
(5.9)	$(N-2)\sum_{f\in\mathscr{F}}\prod_{s\in S_f} C_s$
(5.10)	$\lvert E\rvert\sum_{f\in\mathscr{F}}\prod_{s\in S_f} C_s$
(5.11)	$(N-2)\lvert E\rvert\sum_{f\in\mathscr{F}}\prod_{s\in S_f} C_s$
(5.12)	$\sum_{f\in\mathscr{F}}\prod_{s\in S_f} C_s$
(5.13)	$N\sum_{f\in\mathscr{F}}\lvert S_f-1\rvert\prod_{s\in S_f} C_s$
(5.14)	$\sum_{f\in\mathscr{F}}\lvert S_f-1\rvert\prod_{s\in S_f} C_s$
(5.15)	$N\sum_{s\in S} C_s$
(5.16)	$2N(N-1)\lvert E\rvert\lvert S\rvert$
(5.19)	$N(N-1)\lvert S\rvert$
(5.20)	$N(N-1)\lvert S\rvert$
(5.21)	$N(N-1)\lvert S\rvert$
(5.22)	$2N(N-1)\lvert S\rvert$
(5.23)	$2N(N-1)\lvert S\rvert$

$$d \leq 2N^2+N^2+\left[N+1+N^2\left(N-1\right)\right]C^{|S|}+(N+1)C|S| \\ +(N+1)|S|C^{|S|}+N(N-1)|S|(7+2N^2) \quad \text{(A.6)}$$

Neglecting smaller terms and constants in (A.6):

$$3N^2+N^3C^{|S|}+N|S|C^{|S|}+|S|N^4 \quad \text{(A.7)}$$

Further simplifying and neglecting smaller terms, (A.7) grows as:

$$O(N|S|C^{|S|}, |S|N^4) \tag{A.8}$$

Using (A.4) and (A.8), the complexity of ILP model is:

$$O(256^{\max(N^2C^{|S|}, |S|N^4)} \cdot \max(N|S|C^{|S|}, |S|N^4)) \tag{A.9}$$

In case there is only one state variable required by all the flows, (A.9) simplifies below:

$$O(256^{N^4}N^4) \tag{A.10}$$

References

[1] D. Kreutz, F. M. V. Ramos, P. Esteves Veríssimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[2] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Computer Communication Review*, 38(2):69–74, March 2008.

[3] ONOS website. https://wiki.onosproject.org. Accessed: 17-12-2018.

[4] OpenDaylight website. https://www.opendaylight.org. Accessed: 17-12-2018.

[5] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241, RFC Editor, June 2011.

[6] P. Bosshart, G. Gibb, H. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM Computer Communication Review*, 2013.

[7] M. Bonola, R. Bifulco, L. Petrucci, S. Pontarelli, A. Tulumello, and G. Bianchi. Implementing advanced network functions for datacenters with stateful programmable data planes. In *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6, June 2017.

[8] M. Agiwal, A. Roy, and N. Saxena. Next Generation 5G Wireless Networks: A Comprehensive Survey. *IEEE Communications Surveys Tutorials*, 18(3):1617–1655, 2016.

[9] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider. NFV and SDN—Key Technology Enablers for 5G Networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478, November 2017.

[10] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262, 2016.

[11] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker. SNAP: Stateful Network-Wide Abstractions for Packet Processing. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 29–43, 2016.

[12] A. Panday, C. Scotty, A. Ghodsiy, T. Koponen, and S. Shenker. CAP for networks. In *HotSDN*, pages 91–96. ACM, 2013.

[13] S. Vissicchio, L. Vanbever, and O. Bonaventure. Opportunities and research challenges of hybrid software defined networks. *ACM SIGCOMM Computer Communication Review*, 44(2):70–75, April 2014.

[14] ONOS 1.4 Wiki. https://wiki.onosproject.org/display/ONOS14/Wiki+Home. Accessed: 17-12-2018.

[15] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv. Control plane of software defined networks: A survey. *Computer Communications*, 67:1 – 10, 2015.

[16] P. Bailis and A. Ghodsi. Eventual Consistency Today: Limitations, Extensions, and Beyond. *Queue*, 11(3):20:20–20:32, March 2013.

[17] E. Brewer. CAP twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, February 2012.

[18] Downloads - ONOS - Wiki. https://wiki.onosproject.org/display/ONOS/Downloads. Accessed: 17-12-2018.

[19] D. Ongaro and J. Ousterhout. In Search of an Understandable Consensus Algorithm. In *USENIX Annual Technical Conference*, pages 305–319, 2014.

[20] Linux Containers. https://linuxcontainers.org. Accessed: 17-12-2018.

[21] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[22] ONOS Distributed Flow Rule Store. https://github.com/opennetworkinglab/onos/blob/onos-1.8/core/store/dist/src/main/java/org/onosproject/store/flow/impl/DistributedFlowRuleStore.java. Accessed: 17-12-2018.

[23] OpenFlow 1.0 (Wire Protocol 0x01) specification. https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf. Accessed: 17-12-2018.

[24] OpenFlow 1.3 (Wire Protocol 0x04) specification. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf. Accessed: 17-12-2018.

[25] M. Kuźniar, P. Perešíni, and D. Kostić. What You Need to Know About SDN Flow Tables. In *Passive and Active Measurement*, pages 347–359, 2015.

[26] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievskiy, M. Zhuy, R. Ramanathany, Y. Iwataz, H. Inouez, T. Hamaz, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, pages 351–364, 2010.

[27] L.Schiff, S. Schmid, and P. Kuznetsov. In-Band Synchronization for Distributed SDN Control Planes. *SIGCOMM Computer Communication Review*, 46(1):37–43, January 2016.

[28] F. A. Botelho, F. M. V. Ramos, D. Kreutz, and A. N. Bessani. On the Feasibility of a Consistent and Fault-Tolerant Data Store for SDNs. In *2013 Second European Workshop on Software Defined Networks*, pages 38–43, October 2013.

[29] K. Phemius, M. Bouet, and J. Leguay. DISCO: Distributed multi-domain SDN controllers. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–4, May 2014.

[30] F. Benamrane, F. J. Ros, and M. B. Mamoun. Synchronisation cost of multi-controller deployments in software-defined networks. *International Journal of High Performance Computing and Networking*, 9(4):291–298, 2016.

[31] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone. Scalability of ONOS Reactive Forwarding Applications in ISP Networks. *Computer Communications*, 102(C):130–138, April 2017.

[32] J. P. Fernandez-Palacios, V. López, B. Cruz, and O. G. de Dios. Elastic Optical Networking: An Operators Perspective. In *2014 The European Conference on Optical Communication (ECOC)*, pages 1–3, 2014.

[33] A. Lord, P. Wright, and A. Mitra. Core Networks in the Flexgrid Era. *Journal of Lightwave Technology*, 33(5):1126–1135, March 2015.

[34] J. Zhang, Y. Ji, M. Song, Y. Zhao, X. Yu, J. Zhang, and B. Mukherjee. Dynamic Traffic Grooming in Sliceable Bandwidth-Variable Transponder-Enabled Elastic Optical Networks. *Journal of Lightwave Technology*, 33(1):183–191, January 2015.

[35] ITU-T. G.694.1: Spectral grids for WDM applications: DWDM frequency grid. https://www.itu.int/rec/t-rec-g.694.1, 2012. Accessed: 17-12-2018.

[36] J. L. Vizcaíno, Y. Ye, V. López, F. Jiménez, R. Duque, and P. M. Krummrich. Cost evaluation for flexible-grid optical networks. In *2012 IEEE Globecom Workshops*, pages 358–363, December 2012.

[37] X. Yu, M. Tornatore, Ming Xia, Yongli Zhao, Jie Zhang, and B. Mukherjee. Brown-Field Migration from Fixed Grid to Flexible Grid in Optical Networks. In *Optical Fiber Communication Conference*, page W1I.4, 2015.

[38] V. López, O. González de Dios, L. M. Contreras, J. Foster, H. Silva, L. Blair, J. Marsella, T. Szyrkowiec, A. Autenrieth, C. Liou, A. Sasdasivarao, S. Syed, J. Sun, B. Rao, F. Zhang, and J. P. Fernández-Palacios. Demonstration of SDN Orchestration in Optical Multi-Vendor Scenarios. In *Optical Fiber Communication Conference*, page Th2A.41, 2015.

[39] M. Cantono, R. Gaudino, and V. Curri. Potentialities and criticalities of flexible-rate transponders in dwdm networks: A statistical approach. *IEEE/OSA Journal of Optical Communications and Networking*, 8(7):A76–A85, July 2016.

[40] A. Giorgetti, F. Paolucci, F. Cugini, and P. Castoldi. Dynamic restoration with GMPLS and SDN control plane in elastic optical networks [Invited]. *IEEE/OSA Journal of Optical Communications and Networking*, 7(2):A174–A182, February 2015.

[41] M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, October 2010.

[42] M. Dallaglio, N. Sambo, J. Akhtar, F. Cugini, and P. Castoldi. YANG Model and NETCONF Protocol for Control and Management of Elastic Optical Networks. In *Optical Fiber Communication Conference*, page W3F.5, 2016.

[43] J. Akhtar. YANG modeling of network elements for the management and monitoring of Elastic Optical Networks. In *IEEE International Conference on Telecommunications and Photonics*, pages 1–5, December 2015.

[44] M. Dallaglio, N. Sambo, F. Cugini, and P. Castoldi. Management of sliceable transponder with NETCONF and YANG. In *International Conference on Optical Network Design and Modeling*, pages 1–6, May 2016.

[45] M. Dallaglio, N. Sambo, F. Cugini, and P. Castoldi. Pre-programming resilience schemes upon failure through NETCONF and YANG. In *Optical Fiber Communication Conference*, page W1D.3, 2017.

[46] Open ROADM Multi-Source Agreement. http://www.openroadm.org. Accessed: 17-12-2018.

[47] B. C. Chatterjee, N. Sarma, and E. Oki. Routing and Spectrum Allocation in Elastic Optical Networks: A Tutorial. *IEEE Communications Surveys Tutorials*, 17(3):1776–1800, 2015.

[48] F. Cugini, F. Paolucci, G. Meloni, G. Berrettini, M. Secondini, F. Fresi, N. Sambo, L. Poti, and P. Castoldi. Push-Pull Defragmentation Without Traffic Disruption in Flexible Grid Optical Networks. *Journal of Lightwave Technology*, 31(1):125–133, January 2013.

[49] R. Proietti, C. Qin, B. Guan, Y. Yin, R. P. Scott, R. Yu, and S. J. B. Yoo. Rapid and complete hitless defragmentation method using a coherent RX LO with fast wavelength tracking in elastic optical networks. *Optics Express*, 20(24):26958–26968, November 2012.

[50] R. Wang and B. Mukherjee. Provisioning in Elastic Optical Networks with Non-Disruptive Defragmentation. *Journal of Lightwave Technology*, 31(15):2491–2500, August 2013.

[51] S. Ba, B. C. Chatterjee, S. Okamoto, N. Yamanaka, A. Fumagalli, and E. Oki. Route Partitioning Scheme for Elastic Optical Networks with Hitless Defragmentation. *IEEE/OSA Journal of Optical Communications and Networking*, 8(6):356–370, June 2016.

[52] T. Mizrahi and Y. Moses. Time4: Time for SDN. *IEEE Transactions on Network and Service Management*, 13(3):433–446, September 2016.

[53] P. Megyesi, A. Botta, G. Aceto, A. Pescapé, and S. Molnár. Challenges and solution for measuring available bandwidth in software defined networks. *Computer Communications*, 99:48–61, February 2017.

[54] Y. Li, N. Hua, Y. Song, S. Li, and X. Zheng. Fast Lightpath Hopping Enabled by Time Synchronization for Optical Network Security. *IEEE Communication Letters*, 20(1):101–104, January 2015.

[55] A. S. Muqaddas, M. G. Alabarce, P. Giaccone, and A. Bianco. Exploiting Time-Synchronized Operations in Software-defined Elastic Optical Networks. In *Optical Fiber Communication Conference*, page W4J.6, March 2017.

[56] A. Bravalheri, M. G. Alabarce, A. S. Muqaddas, P. Giaccone, and A. Bianco. Experimental validation of time-synchronized operations for software-defined elastic optical networks. *IEEE/OSA Journal of Optical Communications and Networking*, 10(1):A51–A59, January 2018.

[57] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209, December 2001.

[58] T. Mizrahi and Y. Moses. Time Capability in NETCONF. RFC 7758, RFC Editor, February 2016.

[59] M. Garrich, A. Bravalheri, M. Magalhães, M. Svolenski, Xue Wang, Y. Fei, A. Fumagalli, D. Careglio, J. Solé-Pareta, and J. Oliveira. Demonstration of dynamic traffic allocation in an SDN-enabled metropolitan optical network test-bed. In *International Conference on Optical Network Design and Modeling*, pages 1–6, May 2016.

[60] ONF. OpenFlow Switch Specification, Version 1.5.2 (Wire Protocol 0x06). Technical report, 2015.

[61] A. Shakeri, X. Wang, M. Razo, A. Fumagalli, M. G. Alabarce, E. Oki, and N. Yamanaka. Estimating the effect of Wavelength Selective Switch latency on optical flow switching performance. In *IEEE International Conference on High Performance Switching and Routing*, pages 1–6, June 2017.

[62] T. Mizrahi and Y. Moses. ReversePTP: A Software Defined Networking Approach to Clock Synchronization. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 203–204, 2014.

[63] Calient MEMS-based optical switches. http://www.calient.net. Accessed: 17-12-2018.

[64] Polatis MEMS-based optical switches. http://www.polatis.com. Accessed: 17-12-2018.

[65] A. Bierman, M. Bjorklund, and K. Watsen. Restconf protocol. RFC 8040, RFC Editor, January 2017.

[66] Y. Fei, A. Fumagalli, M. Garrich, B. Sarti, U. Moura, N. G. González, and J. Oliveira. Estimating EDFA output power with an efficient numerical modeling framework. In *2015 IEEE International Conference on Communications (ICC)*, pages 5222–5227, June 2015.

[67] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for Network Update. In *SIGCOMM'12*, pages 323–334, August 2012.

[68] T. Mizrahi, E. Saat, and Y. Moses. Timed Consistent Network Updates. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pages 21:1–21:14, 2015.

[69] M. Zhang, C. You, and Z. Zhu. On the Parallelization of Spectrum Defragmentation Reconfigurations in Elastic Optical Networks. *IEEE/ACM Transactions on Networking*, 24(5):2819–2833, October 2016.

[70] NGMN Alliance. NGMN 5G white paper. Technical report, 2015.

[71] 5GPPP. 5G Vision. https://5g-ppp.eu/wp-content/uploads/2015/02/5G-Vision-Brochure-v1.pdf. Accessed: 01-10-2018.

[72] ETSI. ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration" ETSI GS NFV-MAN 001 (V1.1.1). Technical report, 2014.

[73] ETSI OSM. OSM. https://osm.etsi.org. Accessed: 01-10-2018.

[74] Open Baton community. Open Baton: An extensible and customizable NFV MANO-compliant. http://openbaton.github.io. Accessed: 01-10-2018.

[75] H. Karl, S. Dräxler, M. Peuster, A. Galis, M. Bredel, A. Ramos, J. Martrat, M. S. Siddiqui, S. van Rossem, W. Tavernier, and G. Xilouris. DevOps for network function virtualisation: an architectural approach. *Transactions on Emerging Telecommunications Technologies*, 27(9):1206–1215.

[76] ONAP Community. ONAP Architecture Overview. http://www.onap.org. Accessed: 01-10-2018.

[77] C. J. Bernardos, B. P. Gerö, M. Di Girolamo, A. Kern, B. Martini, and I. Vaishnavi. 5GEx: realising a Europe-wide multi-domain framework for software-defined infrastructures. *Transactions on Emerging Telecommunications Technologies*, 27(9):1271–1280, 2016.

[78] A. Francescon, G. Baggio, R. Fedrizzi, E. Orsini, and R. Riggio. X-MANO: An open-source platform for cross-domain management and orchestration. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–6, 2017.

[79] D. Gkounis, N. Uniyal, A. S. Muqaddas, R. Nejabati, and D. Simeonidou. Demonstration of the 5GUK Exchange: A Lightweight Platform for Dynamic End-to-End Orchestration of Softwarized 5G Networks. In *European Conference on Optical Communication (ECOC)*, page TuDS.14, 2018.

[80] Corsa DP2100. https://www.corsa.com/products/dp2100/. Accessed: 20-01-2019.

[81] CirrOS. CirrOS in Launchpad. https://launchpad.net/cirros. Accessed: 01-10-2018.

[82] J. G. Herrera and J. F. Botero. Resource Allocation in NFV: A Comprehensive Survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[83] Cisco. Cisco Global Cloud Index: Forecast and Methodology, 2016-2021. Technical report, 2018.

[84] Metro-Haul EU Project. https://metro-haul.eu. Accessed: 17-12-2018.

[85] D. King, A. Farrel, and N. Georgalas. The role of SDN and NFV for flexible optical networks: Current status, challenges and opportunities. In *2015 17th International Conference on Transparent Optical Networks (ICTON)*, pages 1–6, July 2015.

[86] R. Casellas, R. Martìnez, R. Vilalta, and R. Muñoz. Control, Management, and Orchestration of Optical Networks: Evolution, Trends, and Challenges. *Journal of Lightwave Technology*, 36(7):1390–1402, April 2018.

[87] V. Lopez, R. Vilalta, V. Uceda, A. Mayoral, R. Casellas, R. Martìnez, R. Muñoz, and J. P. Fernandez Palacios. Transport API: A Solution for SDN in Carriers Networks. In *ECOC 2016; 42nd European Conference on Optical Communication*, pages 1–3, 2016.

[88] OIF-ONF. SDN Transport API Interoperability Demonstration. https://www.opennetworking.org/wp-content/uploads/2017/02/OIF-ONF-2016-SDN-T-API-Interop-Demo-whitepaper.pdf, 2017. Accessed: 17-12-2018.

[89] ONF. Open Disaggregated Transport Neworks ODTN project. https://wiki.onosproject.org/display/ODTN/ODTN. Accessed: 17-12-2018.

[90] ETSI. NFV; Ecosystem; Report on SDN Usage in NFV Architectural Framework - ETSI GS NFV-EVE 005 (V1.1.1), 2015.

[91] ONF. TAPI Reference Implementation). https://github.com/OpenNetworkingFoundation/TAPI. Accessed: 17-12-2018.

[92] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *IEEE Communications Magazine*, 51(2):136–141, 2013.

[93] A. Bianco, P. Giaccone, S. Kelki, N. M. Campos, S. Traverso, and T. Zhang. On-the-fly traffic classification and control with a stateful SDN approach. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2017.

[94] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan. Measuring Control Plane Latency in SDN-enabled Switches. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pages 25:1–25:6, 2015.

[95] C. Kim, P. Bhide, E. Doe, H. Holbrook, A. Ghanwani, D. Daly, M. Hira, and B. Davie. In-band Network Telemetry (INT), 2016.

[96] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, and G. Bianchi. LODGE: LOcal Decisions on Global statEs in programmable data planes. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 257–261, June 2018.

[97] J. McClurg, H. Hojjat, N. Foster, and P. Černý. Event-driven network programming. *SIGPLAN Notices*, 51(6):369–385, June 2016.

[98] C. Monsanto, N. Foster, R. Harrison, and D. Walker. A Compiler and Run-time System for Network Programming Languages. *SIGPLAN Notices*, 47(1):217–230, January 2012.

[99] L. Lamport. Paxos made simple. *ACM Sigact News*, 2001.

[100] K. Birman. The promise, and limitations, of gossip protocols. *ACM SIGOPS Operating Systems Review*, 41(5):8–13, 2007.

[101] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*, pages 386–400. Springer, 2011.

[102] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. Bayou: replicated database services for world-wide applications. In *Proceedings of the 7th workshop on ACM SIGOPS European workshop: Systems support for worldwide applications*, pages 275–280. ACM, 1996.

[103] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer Publishing Company, Incorporated, 2015.

[104] S. E. Schaeffer. Survey: Clustering. *Computer Science Review*, 1(1):27–64, August 2007.

[105] K. Ruddel and A. Raith. Graph partitioning for network problems. In *Joint NZSA ORSNZ Conference*, number 107, pages 1–10, 2013.

[106] CPLEX Optimizer. https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer. Accessed: 17-12-2018.

[107] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *nature*, 393(6684):440, 1998.

[108] B. Gaboune, G. Laporte, and F. Soumis. Expected distances between two uniformly distributed random points in rectangles and rectangular parallelpipeds. *Journal of the Operational Research Society*, 44(5):513–519, 1993.

[109] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[110] S. Luo, H. Yu, and L. Vanbever. Swing State: Consistent Updates for Stateful and Programmable Data Planes. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 115–121, 2017.

[111] G. Bianchi, M. Bonola, A. Capone, and C. Cascone. OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch. *SIGCOMM Computer Communication Review*, 44(2):44–51, April 2014.

[112] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking. Packet Transactions: High-Level Programming for Line-Rate Switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 15–28, 2016.

[113] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé. NetPaxos: Consensus at Network Speed. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 5:1–5:7, 2015.

[114] Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, January 1984.