UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RAFAEL DA FONTE LOPES DA SILVA

A Study on the Effects of Distribution of the SDN Control Plane in Wide Area Networks

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engineering

Advisor: Prof. Dr. Marinho Pilla Barcellos

Porto Alegre
December 2015

"Life can only be understood backwards,
but it must be lived forwards."
— Soren Kierkegaard

" 'I wish life was not so short', *he thought.*
'Languages take such a time, and so do all
the things one wants to know about'."
— J.R.R. Tolkien

ACKNOWLEDGMENTS

I'd like to use this space to thank everyone who had a contribution not only to this work, but also to the whole process that led me up to this moment in my journey, and remind them how important their influence will be to get me forward from now on.

First, I want to acknowledge the importance of my family in my life. They've always supported my every decision over these years, and provided me with everything I could possibly need to reach my goals. More than anything, they have always been around when I'd need them, and I'll be forever grateful for that.

Second, I'd like to thank all the professors that helped me along the way. Each and everyone of you who showed up early in the morning for class with enthusiasm and will to teach allowed me to, little by little, understand more of the world, and also stimulated my interest in exploring its wonders.

Third, I believe it is important to acknowledge the contribution that many friends provided to growth – especially the ones from the networking laboratory. It was amazing to see how, despite the constant "tension" around us, we always managed to get together to help each other when needed. I've never witnessed such great examples friendship and selflessness in a group before, and I believe that it only helped making us stronger to work as a team.

Last, but not the least, I'd like to specifically thank my advisor. More than ever I can now see that our endless discussions always targeted the best for everyone in the laboratory and for myself. Despite eventual issues along the way, he always strives to get the best of me to show up in every aspect of the research process. Thank you for hearing me whenever I had doubts, and for always contributing with thoughtful and highly qualified suggestions to further improve our work. Also, thank you for always believing in my capacity.

ABSTRACT

The move towards an SDN-driven network presents a new set of challenges to network architects. The simplicity of having a centralized entity coordinating state updates among switching elements comes along with concerns such as overload and delayed network view in the control plane. Those issues become especially relevant in the scenario of Wide Area Networks. One way to attempt to mitigate them is to allow a relaxation of the concept of a centralized controller, by leveraging the notion of physical distribution of control instances. Such control plane design might reduce latency from each control instance to its corresponding devices, resulting in better response to local network events and improved overall network performance. However, the distribution of control itself comes with its own caveats, mostly related to the need to synchronize the distributed instances. In this context, we present the first thorough attempt at identifying and quantifying how control plane distribution design choices affect network operation, specifically in WAN scenarios. We perform an evaluation campaign using emulations with a real global network topology, as well as synthetic one, in order to identify important trade-offs and compare distributed and centralized control plane strategies. We also demonstrate that a geographically distributed controller that relies on constant state synchronization reacts slowly to network events when compared to a simpler geographically centralized solution.

Keywords: Software-defined networking. network control plane. wide area networks. distributed systems.

Um estudo sobre os Efeitos da Distribuição do Plano de Controle de Redes Definidas por Software em Redes de Longa Distância

RESUMO

A transição para Redes Definidas por Software apresenta um novo conjunto de desfafios para arquitetos de rede. A simplicidade de se ter uma entidade centralizada que coordena atualizações de estado entre elementos comutadores traz consigo preocupações como sobrecarga e visão atrasada da rede pelo plano de controle. Estas questões se tornam relevantes especialmente no cenário de Redes de Longa Distância. Uma alternativa para se tentar mitigá-las é permitir um relaxamento do conceito de um controlador centralizado, utilizando-se da noção de uma distribuição física de instâncias de controle. Esse estilo de plano de controle tem o potencial de reduzir a latência de cada instância de controle para com seus dispositivos correspondentes, resultando em melhor resposta para eventos de rede locais e melhor performance de rede em geral. Porém, a distribuição de controle em geral traz à tona desafios próprios, principalmente pela necessidade de sincronização entre as instâncias distribuídas. Neste contexto, nós apresentamos uma primeira tentativa de identificar e quantificar como escolhas de projeto do plano de controle afetam a operação da rede, especificamente em cenários de Redes de Longa Distância. Realizamos uma avaliação em que emulamos tanto uma topologia de rede global como com uma sintética, de modo a identificar importantes relações de custo-benefício e a comparar estratégias de plano de controle distribuído e centralizado. Demonstramos ainda que um controlador geograficamente distribuído que depende de sincronização constante de estado reage lentamente a eventos de rede quando comparado a uma solução simples de controlador geograficamente centralizado.

Palavras-chave: redes definidas por software, plano de controle da rede, redes de longa distância, sistemas distribuídos.

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Program Interface
BGP	Border Gateway Protocol
CDF	Cumulative Distribution Function
GNV	Global Network View
ICMP	Internet Control Message Protocol
IP	Internet Protocol
LAN	Local Area Network
LLDP	Link-Layer Discovery Protocol
MPLS	Multi-Protocol Label Switching
NOS	Network Operating System
NIB	Network Information Base
OVS	Open vSwitch
ONOS	Open Network Operating System
OSPF	Open-Shortest Path First
RTT	Round-Trip Time
SDN	Software-Defined Networking
STP	Spanning Tree Protocol
UDP	User Datagram Protocol
VLAN	Virtual LAN
VM	Virtual Machine
WAN	Wide Area Network

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The increasing complexity of Internet service infrastructures called for new mechanisms to provision and manage resources. The concept of Software-Defined Networking (SDN) emerged within this context, seeking to allow more effective ways of programming computer networks. An important characteristic of this paradigm is the separation between the control and the data plane of the network. With this separation, the control plane becomes a logically centralized entity. It is responsible for managing the underlying datapath infrastructure, using an API such as OpenFlow (N. McKeown et al., 2008), and also to provide an abstract view of that infrastructure to control applications. Since the introduction of SDN as a new architectural paradigm, the networking community has been encouraged to revisit network control from the perspective of the abstractions it enables, producing very interesting results along the way (KREUTZ et al., 2015).

The study of Wide Area Network SDN deployments is particularly interesting, in part due to their distinctive requirements over more traditional SDN deployments such as in datacenters. Typical WAN deployments face challenges with low resource utilization as a consequence of policies to mitigate failures and loss of data. Leveraging the principles of SDN enables operators to overcome these issues, increasing the overall efficiency of these networks and reducing infrastructure costs (S. Jain et al., 2013; HONG et al., 2013).

The logical centralization of the SDN control plane provides a clean abstraction to network designers. It also introduces several challenges, such as how to deal with possible scalability limitations. The body of work on the design of SDN control plane constitutes itself a quite diverse research area. Related work on the design of SDN control planes can be separated into three main categories: controller roles and interfaces (Koponen, T. et al., 2010; TOOTOONCHIAN; GANJALI, 2010; YEGANEH; GANJALI, 2012), controller positioning (SCHMID; SUOMELA, 2013; ZHANG; BEHESHTI; TATIPAMULA, 2011; HELLER; SHERWOOD; MCKEOWN, 2012; MULLER et al., 2014) and distribution trade-offs (LEVIN et al., 2012; YEGANEH; GANJALI, 2014; AKELLA; KRISHNAMURTHY, 2014).

The commonly proposed solution for scaling and increasing resilience of the SDN control plane is the physical separation of the control plane among different *control instances*. In the case of an SDN WAN, control instances could be *geographically separated*, in contrast to deployments using a single control instance or multiple clustered ones. It is surprising, therefore, that the effects of this geographical distribution of control have not been investigated in the SDN literature. We envisage as main potential challenges variable responsiveness to global network updates, delays introduced by controller synchronization, and control traffic overhead.

This study presents an initial attempt at understanding such effects. By devising a testbed tailored to SDN WAN scenarios, and using current, production level software (such as the ONOS distributed controller and the Open vSwitch virtual switch), we focus on answering two fundamental research questions: *how does the inherent propagation latency of WAN networks affect general communication in a distributed control plane setup?* and *how does the control plane distribution scheme impact on intensive data plane communication?* Our results indicate that the need for frequent synchronization significantly harms the performance of geographically distributed control planes in responding to network events, resulting high latency to reactions.

This study is organized as follows: Chapter 2 discusses related work on SDN control plane design, with focus on SDN WAN deployments. Chapter 3 lays the foundation to our study, and also describes our testbed and experimental methodology. Chapter 4 presents and discusses our experimental results, focusing on answering our research questions. Finally, Chapter 5 concludes this work and presents possibilities for future work.

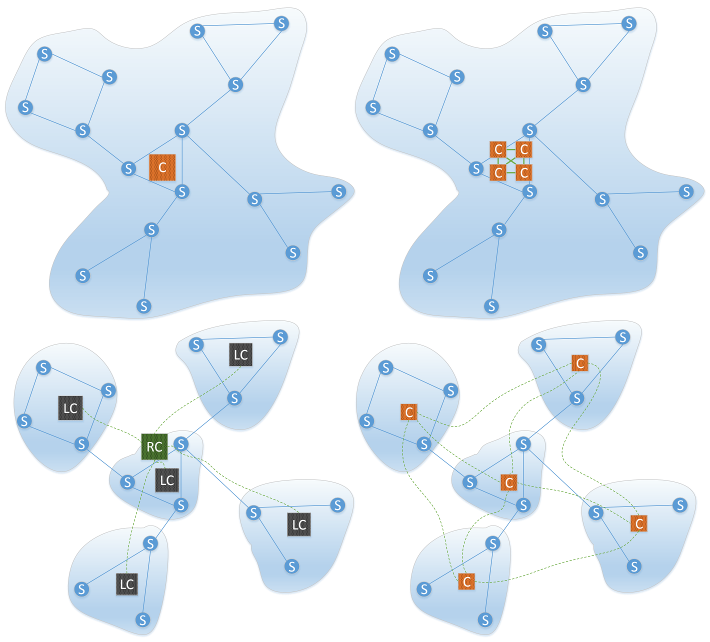
2 LITERATURE ON SDN CONTROL PLANE DESIGN

The separation between control and data planes in the SDN paradigm is often advocated as being the key enabler for easing the development of control applications in an organized, modular way, representing a great shift from traditional network protocol design. A core element of this separation is the aggregation of the control plane as a logically centralized entity. This concept relates to the idea of aggregating network state inside the Network Operating System – NOS (sometimes in the form of a Global Network View abstract graph – GNV), while conveying it to network applications in a simple, seemingly centralized manner. This way, common networking functionality can be implemented at a central entity and be provided as simple abstractions to network applications.

It is challenging to implement a logically centralized controller in real networking environments. Most deployments require the control plane to handle a large amount of interactions, due either to intensive network behavior or to a large quantity of network nodes to manage. The range of use cases also tend to present diverse network requirements. Hence, proposals in SDN control plane design have moved on from simpler, single machine approaches, to scalable distributed systems. This allowed SDN to meet the performance required in these scenarios and leverage its adoption in production environments. Figure 2.1 illustrates the main distributed control plane structuring approaches, in which we focus our attention. The notion of distribution we discuss in our analysis relates to the *positioning* of the control instances in the WAN network – either *physically separated* (i.e. distributed SDN WAN control plane) or *physically co-located* (i.e. centralized SDN WAN control plane).

Some of these design approaches will be presented in the forthcoming sections. We summarize their features in table 2.1, establishing a comparison between their advantages and drawbacks. The rest of the chapter is organized as follows: Section 2.1 presents the main approaches to SDN control plane organization. Section 2.2 identifies general trade-offs in the design of control planes. Section 2.3 concludes this chapter by describing the attempts at deploying SDN in WAN settings that were identified.

Figure 2.1 – The main approaches to SDN control plane distribution design. Top: classic centralized (left) and physically centralized clustered (right) control planes. Bottom: hiearchical (left) and physically distributed (right) control planes.



Source: by author.

2.1 Control Plane Organization

In this section, we briefly discuss the main design approaches to SDN control plane organization. Section 2.1.1 presents the initial SDN control plane designs, which consider only single machine centralized implementations and parallelization optimizations. Section 2.1.2 presents the hierarchical approach to controller decentralization, discussing some of its advantages and drawbacks in terms of controller load and visibility. Finally, Section 2.1.3 concludes by discussing the design of fully distributed control planes, where control instances have very similar roles in the network management process.

2.1.1 Classic Centralized Control Plane

Logical centralization of the network control plays a central role in an SDN architecture. Initial efforts towards implementing such an architecture focused on its feasibility in real world scenarios (e.g. campus networks) and on identifying benefits over traditional approaches. Initial designs leveraged the use of single controller deployments (GUDE et al., 2008), relying on the now-standard OpenFlow protocol (N. McKeown et al., 2008). Later, other designs have been developed and released (e.g. (MCCAULEY, 2014; ERICKSON, 2013; CAI; COX; NG, 2010; Ryu SDN Framework Community, 2015)). However, most of them do not provide a very well defined northbound API, nor attempt to shield the application developer from dealing with certain low-level mechanisms of OpenFlow.

Recent work provides interesting conclusions in terms of centralized controller performance. (TOOTOONCHIAN et al., 2012) show that, with minor code optimizations and a redesign of the controller structure to support multithreaded parallelism, one can drastically improve throughput and reduce latency on a centralized, single machine controller. Via these optimizations, it has been possible to implement controllers capable to process up to 12 million *packet-in* messages per second (ERICKSON, 2013).

The research community often argues that, despite the advancements in single controller processing capabilities, it is still not enough to meet the performance, scalability and resiliency requirements of large scale production environments (Koponen, T. et al., 2010; TOOTOONCHIAN; GANJALI, 2010; KREUTZ et al., 2015). The intrinsic limitations of this type of design, such as increased latency to forwarding devices in large networks and difficulty in handling large network state, have motivated the development of distributed control plane designs.

2.1.2 Hierarchically Distributed Control Plane

Hierarchical control planes are developed around the idea that control instances should have different roles in the control process. **Kandoo** (YEGANEH; GANJALI, 2012) explicitly separates control among two different layers of controllers. The top layer is composed of a *root controller*. It is responsible for maintaining global network policies and pushing those down to the bottom layer controllers. The bottom layer is comprised of *local controllers*. These controllers directly configure the network forwarding devices, and share their perceived network state with the root controller. This approach has the direct benefit of simplifying the presentation of the Global Network View to control applications at the root controller. However, it requires the development and maintenance of one type of controller for each control plane layer. The root controller introduces concerns as it represents a single point of failure, since it is required to deal with constant network updates. Moreover, it has to operate on an *eventually consistent* network state, since it can only perceive state that is reported by local controllers. Thus, this design is somewhat limited in applicability, being more favorable in low latency environments like datacenters, where traffic locality can be well explored in most cases.

(YU et al., 2010) presents **Difane**, a solution for network control in which so-called *Authority Switches* are assigned the role of forwarding rule *caches* – much in line with what the local controllers illustrated previously perform in Kandoo. The objective of this approach is to delegate some of the control concerns to the data plane, thus limiting the load imposed over the network controller. Similarly, (A. R. Curtis et al., 2011) presents a hierarchical approach to control delegation in which forwarding devices handle most of the data traffic without controller intervention. Specific *significant* flows are identified according to certain traffic features, and are still handled by the controller directly. This way, datapath flow tables should be handled more efficiently, and flow rules suffer far less from flow setup overhead.

Despite the advantages of these approaches to scaling the network to larger sizes, they achieve their results by breaking the principle of separation between control and data planes, as they allow some of the control tasks to be handled at the data plane. Hence, designs that consider delegation of control to forwarding devices must be carefully planned as they could hide important state information from control applications. Also, they favor proactive network control by design, and thus could prevent the network controller from taking faster, near globally optimal actions for certain types of network events.

Another interesting hierarchical control approach is to leverage the notion of a *recursive SDN design*, which is a rather recent effort in this area (MCCAULEY et al., 2015; KWAK et al., 2015; DAI et al., 2014). **R-SDN** (MCCAULEY et al., 2015), for instance, presents an interesting solution around such design. The basic insight behind it is that, for most deployment cases, control plane operations at the different levels of network control – subnet protocols, site protocols, region protocols and WAN protocols – can be seen as very similar tasks, but with different requirements. In traditional network approaches, these different levels of control would be typically implemented by different protocols (e.g. STP for subnets, OSPF for sites, BGP for peering autonomous systems in the Internet). RSDN aims at following this same type of structure, and also re-uses control plane implementations for these different levels of control through aggregation with the so-called *Logical Crossbars*. This approach has the potential of scaling more easily to very large networks such as carrier networks, where it should provide good recovery properties for locally-based failures.

2.1.3 Fully Distributed Control Plane

Fully distributed control planes are composed of control instances with very similar roles and capabilities, constituting a horizontal approach to control plane organization. Coordination among instances is essential, as they need to properly configure forwarding devices in order to achieve global network goals/policies, while at the same time ensuring good performance in the network.

Works such as (Koponen, T. et al., 2010; TOOTOONCHIAN; GANJALI, 2010) provided the initial insights on how SDN control plane distribution should be performed. They both argue that simpler, centralized controller deployments can severely suffer from delayed responsiveness by the controller as networks grow in number of nodes – implying in controller overload – and in diameter (as control messages need to traverse longer paths to connect controllers and forwarding devices). A common concern over such distribution is regarded to what the best trade between application agnosticism of the underlying distribution scheme (i.e. simplicity of implementation) and performance of control operations (e.g. quicker reaction to network events) is. This type of analysis is performed in (LEVIN et al., 2012), in which the results suggest that agnosticism to the underlying control distribution and staleness of network view can severely impact network performance.

Onix (Koponen, T. et al., 2010) is a control platform designed to provide scale-out capabilities to network control while allowing great flexibility in terms of network state choices for control application developers. The central abstraction Onix provides for applications is the *Network Information Base* (NIB). This structure presents the state in the form of a GNV graph. Since network control can be distributed, the platform allows designers to choose the level of state consistency they require for their applications. For instance, if they wish to impose strong consistency – at the penalty of possible delay in the reaction to network events – they can implement a *transactional database store* in the NIB. If, however, such constraint can be relaxed (e.g. when dealing only with routing in the locality of each instance), they can also employ an *eventually consistent store*, such as a *Distributed Hash Table*. Despite introducing further complexity into network applications, this approach also enables designers to decide on trade-offs regarding features such as consitency, durability and scalability.

Hyperflow (TOOTOONCHIAN; GANJALI, 2010) is another approach to control plane distribution. It works as an application that runs over the NOX controller (GUDE et al., 2008). In order to provide faster response times, each Hyperflow controller instance is positioned in a different region of the network, and all important network state updates are instantly shared among control instances. This way, according to the authors, it is possible to avoid spikes in latency when certain global policies are applied, since instances don't need to perform special requests among them. When only local network state is involved, controllers can be closer to targets and take local decisions immediately. Applications that run on Hyperflow-enabled controllers can manipulate network state as if they were running centrally, but still require some modifications to adapt to the abstractions provided by the Hyperflow application. Notwithstanding, this freedom in control application design and the overall push/subscribe structure of the state synchronization employed by Hyperflow greatly limit the amount of network events that can be handled by the control plane. It was identified that excessive events in the network (around 1000 events per second) greatly diminish control plane responsiveness performance, being prohibitively large for proper network operation. Its current design cannot scale the control task well when the events that trigger controller synchronization are a function of the total number of flows in the network. Thus, in order to mitigate any undesirable effects, improvements over Hyperflow need to consider modifications on the network control mechanisms. For instance, state aggregation (to minimize control message size), less frequent state information updates and proactive control applications can greatly improve the performance of Hyperflow, eventually at the cost of a less frequently consistent network view.

ONOS (P. Berde et al., 2014) is an open-source network controller platform (i.e. NOS). The design goals behind ONOS include high throughput, low latency in event processing, capability of handling a large GNV and high availability. Different third-party tools were leveraged to simplify its construction. For instance, Zookeeper (HUNT et al., 2010) was integrated to manage mastership relations among forwarding devices and control instances, whereas Cassandra (LAKSHMAN; MALIK, 2010) facilitates controller synchronization among control instances. Over time, the project moved to its own custom implementation for many of these modules, and continues to be improved today (ON.LAB, 2015). To react faster to events such as network partitions and swiftly perform the discovery process, ONOS control instances constantly probe the network by sending *LLDP* packets out forwarding device ports. Since ONOS aims at providing scale-out capabilities to meet the requirements of many use cases, with an emphasis on carrier networks, it provides the option of forming *clusters* of ONOS controllers. Each controller in a cluster has the task of managing a subset of the data plane forwarding devices, improving performance in the handling of network events. In order to provide high availability and reliability, forwarding devices are assigned both a master and standby controllers. These properties and the availability of this project as open source led us to adopt ONOS as the control platform in our testing environments, as will be discussed later in this document.

2.2 Trade-offs in SDN Control Plane Design

The literature on SDN includes discussions on different trade-offs involved in the design of the control plane. The separation between control and data planes is often pointed out as a benefit, since it leverages the use of abstractions to allow the evolution of each plane independently. However, it also poses a challenge, due to the inherent dependency between the devices associated with each plane. The works presented below focus on particular aspects of this and other control plane design options, in order to shed light on what the best choices for each particular SDN deployment are, according to its requirements.

In (LEVIN et al., 2012), the authors argue that SDN designers must be careful when dealing with the consistency models employed. Specifically, they compare an *eventually consistent* state design and a *strongly consistent* one. In their study, they identify and analyze important trade-offs related to control application performance and the underlying consistency model when the control of the network is distributed among different controller instances. By experimenting with different versions of a load balancer application, they identify that network functions that require fast responsiveness from the control application cannot be agnostic to the state distribution features of the control plane.

The sequencing of updates on the forwarding state in an SDN network is also an interesting area of research that can directly influence the design of the control plane. In (REITBLATT et al., 2012) the authors provide an extensive analysis on the problem of consistent updates, arguing that the capacity of constructing abstractions provided by the SDN architecture presents an excellent opportunity towards managing such updates. They present a simple model of a general OpenFlow network, over which they construct mechanisms to guarantee consistent updates in the network forwarding state. These mechanisms give the network the interesting ability to transition from a currently employed policy to a final goal policy, while guaranteeing that each packet traversing such network is only processed according to only one of them, and never a mixture of the two. Further works attempt to deal with scenarios that consider a distributed control plane along with concurrency of policy updates (CANINI et al., 2015). They also consider the possibility of failure of control instances during the update process. None of these works on consitent updates, however, provides an extensive study of the impact that the proposed mechanisms have in the operation of a real SDN network deployment. The network would have to deal with challenges regarding flow table occupancy (since forwarding devices would have to keep flow entries for both current and goal policies for a certain period of time) and coordination among controller instances while updates take place in the data plane.

Another important consideration when dealing with geographical distribution of control is to determine the *placement and number of controller instances* to position over the network topology. Authors in (HELLER; SHERWOOD; MCKEOWN, 2012) explored some of the fundamental trade-offs to be considered in this decision, such as those involving controller-to-switch latency and number of control instances required. The authors of (ZHANG; BEHESHTI; TATIPAMULA, 2011), on the other hand, tackled the problem motivated by the resilience issues involved in the split planes feature of the SDN architecture, attempting to guarantee connectivity between forwarding devices with at least one controller instance. (MULLER et al., 2014) further enhances these results, by taking into account a variety of metrics such as controller capacity, device demands and also the quantity of disjoint paths between controller instances and forwarding devices.

Other works revisit some of the fundamental ideas of SDN, seeking the proper ways to tackle the control plane design challenges. In (PANDA et al., 2013), a discussion about the features of distributed control plane SDNs with regards to the *CAP theorem* is made, pointing out that there are certain fundamental trade-offs between availability and the enforceability of certain network policies. In (SCHMID; SUOMELA, 2013) the authors discuss functions that an SDN controller typically implements from the standpoint of *local algorithms*, arguing that the use of such algorithms can potentially result in greater efficiency gains for these functions. The authors of (AKELLA; KRISHNAMURTHY, 2014) discuss some of the current flaws in distributed control plane deployments. They illustrate concerns over the way that SDN networks are structured today and their current reliance on legacy protocols to function properly, having serious consequences on network availability. Fortunately, they present some principles from well-known distributed system protocols that can help SDN overcome these shortcomings. Finally, in (CASADO et al., 2012) the authors discuss SDN limitations that surfaced over the years, motivating an architecture in which principles from MPLS and SDN are combined to construct a very interesting network structure. In such architecture, the core of the network is made of switches that forward packets based on simple labels, whereas the edge implements the complex functionality with distributed control instances and specialized edge switches.

There are also works that attempt to tackle the challenge of control distribution in a theoretical manner. The authors in (MATNI; TANG; DOYLE, 2015) present an interesting taxonomy between classes of network control architectures: distributed control architectures, encompassing both control schemes that work only locally on network management and those that coordinate state globally among themselves; and central control architectures, regarding standard centralized SDN approaches. The authors also argue that there is an inherent trade-off between distributed and centralized approaches, pointing out that while a centralized solution will provide more predictable response times (with higher latency in general), a distributed solution can provide better local response times, while presenting more variable response times.

2.3 SDN WAN deployments

Privately-owned Wide Area Networks today need to be properly provisioned and designed to provide value to businesses. The high costs of maintaining this kind of infrastructure, as well as the need for better means of extracting performance from it, have led Google to develop their B4 network (S. Jain et al., 2013). This network is responsible for providing connectivity between their globally distributed datacenters, thus representing an essential asset to leverage the operation of the services they provide. In order to reach their goals, Google leveraged the fact that they control every aspect of their networks, from applications running in their servers to the Internet-facing edge, thus greatly benefiting from the predictability and configurability properties in the design of their infrastructure.

The core of their solution to increase efficiency was to use a Software-Defined Network architecture for control with a customized implementations of OpenFlow and Onix. This transformation into an SDN WAN led to a series of advantages: (i) link utilization levels in B4 have increased close to 100% – usually, their WAN deployment would operate at 2 to 3 times overprovisioning for their links to minimize packet losses and mitigate failures, incurring in increased operation costs; (ii) the switching fabric could now be built by cheaper, simpler and custom hardware, instead of the usual proprietary, high end expensive WAN routers; and (iii) they could easily iterate over new versions of their central traffic engineering solution, as well as deploy new functionality much faster than if they only used solution comprised of traditional networking protocols.

Google also illustrated their bandwidth enforcement solution – BwE – for enabling efficient utilization of their WAN infrastructure (KUMAR et al., 2015). They present the benefits of their scheme, in which bandwidth control is performed all the way down to the application level in individual servers. Google employed a hierarchical SDN approach to control their networks at different levels of aggregation, going as far as modifying the networking stack of individual machines in order to improve traffic classification and thus properly enforce priority and provide better efficiency. The higher levels of BwE (e.g. Site and Cluster enforcers) are responsible for monitoring aggregate flow statistics. Taking these into account, along with global policy information, they redistribute bandwidth allocation among the lower levels. Bandwidth enforcement is then applied at the kernel of individual machines. This scheme of bandwidth control has also greatly contributed to the performance levels achieved with B4.

The scientific community has also invested on SDN solutions built specifically for the WAN setting. For instance, (GEROLA et al., 2015a; GEROLA et al., 2015b) illustrate ICONA, a network application built for ONOS to scale the network control plane in globally deployed networks with multiple domains. It provides connectivity among different ONOS clusters – each of those, while being comprised of multiple instances, positioned in at strategic points of the network. The authors attempt to explore locality of control for events local to a cluster region, while still allowing inter-region communication in the network as a WAN. While promising, the results presented for tasks such as path rerouting and network discovery show little improvements of using ICONA in these environments over a standard single ONOS cluster approach. The experiments presented in (GEROLA et al., 2015a) were performed using an emulated network environment with *Mininet* (LANTZ; HELLER; MCKEOWN, 2010), along with the *netem* (Linux Foundation, 2015) tools to simulate the global dispersion effects of WANs.

Similarly to the efforts taken by Google with their B4 network, (HONG et al., 2013) present **SWAN**, a centralized SDN solution to increase overall WAN utilization. The scenarios explored by SWAN also comprise inter datacenter WANs. The approach taken with SWAN is to leverage the use of central control to reach global optimal network provisioning and bandwidth allocation. In order to do that, SWAN requires a small amount of free capacity reservation on links to ensure that the reconfiguration processes will not produce congestion in the network. Also, analogous to what is done with B4, SWAN is responsible for negotiating network resource allocations with network services. The centralized nature of the solution allows for better reasoning over such allocation, as the state of the whole network is aggregated at a single entity, which can then reason over shared link allocation and application scheduling. This is in contrast to more traditional distributed solutions such as MPLS Traffic Engineering, which greedily allocate traffic based only on remaining shortest paths with enough capacity in the network.

Table 2.1 – Summary of SDN control plane organizations described in the literature of control plane design.

Organization		Advantages	Drawbacks	Target	Notable examples
Single Machine		Simpler design in general; easier to construct abstractions for control applications; faster prototyping	Poor resilience compared to its distributed counterparts; becomes a bottleneck under high load; higher latency to distant network events	LAN scenarios; control application prototyping and simulation	NOX; NOX-MT; BEACON; Maestro; Ryu
Hierarchically distributed	Specialized control instance roles	Mitigates controller overload; faster response to local events	Root controller constitutes a SPF; different control implementations for the different levels of the hierarchy	Datacenters with locality of traffic	Kandoo
	Devolved control plane	(Presumably) simple adaptation of forwarding devices; faster local response times; mitigates controller overload	Loss of visibility by the controller to certain events; concerns with consistency of updates; devolution breaks the separation of concerns advocated with SDN	Datacenter networks	Difane; DevoFlow
	Recursive SDN	Simpler, replicable controller design; good separation of concerns through levels of aggregation, leading to reduced controller overload; good response to local failures; good scalability potential	Delayed global policy updates; diminished visibility of lower level state updates to higher levels of the hierarchy; difficulty of devising a general solution that can be applied to the different levels of the hierarchy; control network can be cumbersome to manage	Carrier networks; global networks in general	RSDN; RAON; R-SDN
Fully distributed	Physically centralized	Scales to larger numbers of forwarding devices; fast synchronization among control instances; smaller variability in response times due to centralization; can be deployed in a dedicated cluster (easier to maintain)	Increased latency to distant network events; SPF; partitions can cause more impact in network connectivity	Globally-deployed networks (e.g. WAN) and datacenter networks	ONOS; BwE+B4 (custom Onix); SWAN; Onix; Hyperflow
	Physically distributed	Faster response to local events; better resilience to partitions when compared to its centralized counterpart	Higher variability in network response times; issues with delayed synchronization among control instances; need for managing the synchronization traffic	Globally-deployed networks (e.g. WAN)	DISCO; ONOS; ICONA; Onix

Source: by author.

3 CONTROL PLANE DISTRIBUTION STUDY

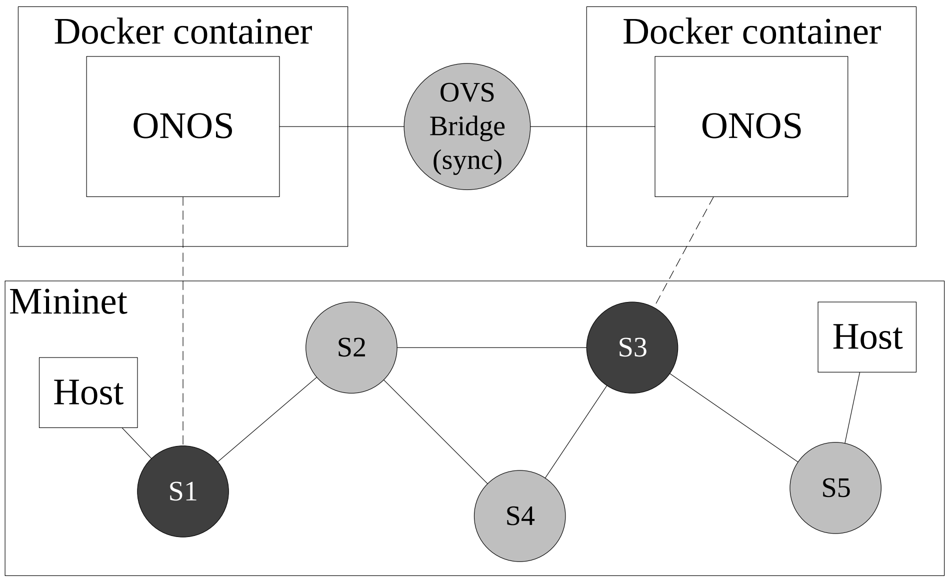
This study presents a novel analysis of SDN control plane distribution trade-offs in Wide Area Network scenarios. The following research questions were fundamental to guide our analysis process of comparing clustering and geographical distribution of network control planes:

- ***How does the inherent propagation latency of WAN networks affect general communication in a distributed control plane setup?*** Since WAN infrastructures present a large maintenance cost overhead, it is of interest of operators to keep utilization levels high (S. Jain et al., 2013; KUMAR et al., 2015; HONG et al., 2013). To that end, guaranteeing fast responsiveness from the control plane to network events and keeping to schedules is of fundamental importance. While longer communication latency can severely affect application performance, variability can limit link utilization efficiency;
- ***How does the control plane scheme adopted impact on intensive data plane communication?*** It is essential to determine the effects on data plane communication when the control plane is under heavy load due to higher utilization demands from applications. The situation becomes even worse in the WAN setting, as the intrinsic link delays can severely affect the visibility of the control plane. This may lead to losses related to compromised control plane responsiveness, thus greatly diminishing efficient link utilization. With this question in mind, we also attempt to understand how the control plane itself behaves in these extreme situations (e.g., what happens to control traffic when demands are higher?).

3.1 Evaluation environment

An SDN WAN testbed was developed in order to compare the centralized and distributed control plane schemes experimentally. In this testbed, the control plane is realized by deploying multiple ONOS instances, whereas the virtual forwarding devices and hosts are instantiated and managed using the Mininet network emulation tool (LANTZ; HELLER; MCKEOWN, 2010). Geographic scale effects of the network under test are made possible by making use of the Linux *netem* (Linux Foundation, 2015) set of traffic control tools (such as queue disciplines), allowing us to set up WAN properties such as link delays with great flexibility. These tools are commonly used in network emulation evaluations (YAN; JIN, 2015; GEROLA et al., 2015b), and their use is supported by Mininet. In order to facilitate deployment of new control instances, we leverage the use of *Docker* containers (Docker, Inc., 2015). These containers are provided with their own isolated process and file system namespaces, and can be easily customized to our needs. For instance, this allows us to make sure that each container only keeps the essential set of tools (e.g. Java runtime environment, ONOS) to minimize system overhead. Also, the architecture of the Docker management system helps enabling the automatization of the experiment process through scripting.

Figure 3.1 – A possible instance of our testbed architecture, illustrating its components. ONOS instances run inside Docker containers, and interact with the underlying data plane infrastructure through a direct link with certain forwarding devices. An OVS bridge is used to allow synchronization among instances.



Source: by author.

The topology employed as reference for our study is extracted from the Abilene Network[1]. It spans 11 nodes throughout the United States territory. We collected topology information from the Internet Topology Zoo repository (KNIGHT et al., 2011). Such information allows us to both set node connectivity and calculate propagation delays in the topology links.

Controller instance positions have been calculated as defined in (MULLER et al., 2014). The methodology described in this work provides the best controller-to-switch mapping in terms of diversity of disjoint paths between the two types of entities, thus favoring network resilience – a frequent argument towards implementing geographical control distribution in WANs. Figure 3.3 illustrates the network topology, along with the resulting switch-to-controller map and

[1]Abilene has been recently upgraded into the Internet2 (The Internet2 Community, 2015), having now one less node in comparison

the instance positions. The best controller placement for this topology with regards to network resilience corresponds to two instances, one located in the Chicago region (node $S1$) and another in the Kansas City region (node $S7$).

3.1.1 Open vSwitch

Open vSwitch is a project that develops and maintains a production quality open source virtual switch implementation. It is compliant with many different networking technologies, such as OpenFlow, IPv4 and STP, and also provides support for many features such as VLAN tunneling and network monitoring via NetFlow and sFlow. It also allows for switching to be performed either in kernel space (enabling performance of switching) or in user space (leveraging flexibility of user space software). The design of OVS aims at network virtualization environments, where it provides connectivity between VMs (Open vSwitch, 2013; PFAFF et al., 2015).

In our experiments, the OVS nodes are configured to communicate with their controllers using the *in-band* mode. This mode allows southbound control traffic (i.e. OpenFlow messages) to be handled by network switches, in contrast with the more common out-of-band scheme. Thus, it does not require a complete separate control network infrastructure to be instantiated. In OVS, this scheme is implemented by making use of high priority flow rules in the switch, which are handled exclusively in the data plane in an L2-switching fashion – thus, the control plane of the network has no control over it. These rules allow for basic network discovery and forwarding functionality for OpenFlow messages, providing a bootstraped environment to allow network discovery and management by the network controller.

3.1.2 Mininet

Mininet (LANTZ; HELLER; MCKEOWN, 2010) is a network emulation tool that allows researchers to perform tests in custom network topologies in a simple manner. It works by virtualizing and isolating multiple network stacks, one for each node in the emulated network (with the exception of network switches – we discuss this matter further ahead), as well as establishing virtual links between their interfaces. The instantiation process is constructed to ease the deployment of network emulations by requiring just a few steps of configuration. Subsystems other than the network stack are shared among the different nodes, such as the process space and the filesystem. Since Mininet does not employ fully-fledged hardware virtualization, it allows for much better resource utilization and performance than in the case of complete virtual machines, which require the use of multiple copies of the system kernel.

To provide connectivity among nodes, Mininet makes use of Open vSwitch switches, in which they are referred to more generally as network bridges. Since Mininet is primarily configured to run on out-of-band scenarios, it leverages the creation of all bridges under the same network stack of the host machine. This allows the OVS bridges to easily reach an external controller through the system's loopback and/or gateway interfaces. This setup, however, does not fit our experiment scenario at all, as it also potentially allows control messages to leak among the datapath bridges and take routes which would be infeasible in real scenarios. Thus, we need to isolate each OVS bridge within its own network stack namespace, in order to ensure that the datapath will behave properly regarding the OpenFlow message exchanges. In order to do that, we leveraged the already isolated network stacks for the Mininet hosts, and deployed the network switches as different instances of OVS, each running on its own host. We make use of a methodology similar to that employed in (Gregory Gee, 2014).

3.1.3 ONOS

ONOS (P. Berde et al., 2014) is a control platform that, similarly to other projects (e.g. OpenDaylight (MEDVED et al., 2014)), attempts to provide a complete, open-source solution for network control. Built on top of Apache Karaf (Apache Software Foundation, 2015b), it allows for a modular composition of network applications that share the same GNV. The platform is responsible for retrieving and constantly updating information extracted from the data plane, and sharing such information with control applications that are built on top of it. Furthermore, it provides operators with the ability to distribute mastership of datapath forwarding devices among multiple separate control instances, allowing the control plane to meet their scale-out needs. In our experiments, we run ONOS with the minimal set of applications required to allow basic control plane operations, such as ARP message management, network discovery (with LLDP packets), control instance clustering (to enable synchronization among control instances) and reactive forwarding (basic forwarding mechanism based on *packet-in* events). The applications we activate – other than the builtin ones – are *org.onosproject.config* (for network configuration), *org.onosproject.drivers* (for certain device drivers), *org.onosproject.fwd* (for the reactive forwarding behavior), *org.onosproject.openflow* (to enable OpenFlow connectivity providers) and *org.onosproject.proxyarp* (for ARP peering).

3.1.4 Environment specifications

The testbed runs on a virtual machine with an Intel(R) Xeon(R) CPU @ 3.50GHz processor, Ubuntu (64 bits) 14.04.2 (kernel 3.16.0-37-generic x86-64) operating system and 6553 MB of RAM. The hypervisor machine runs VirtualBox 4.3.6 to manage the VM on a Windows 7 64-bit operating system. Inside the testbed, we use Open vSwitch 2.3.90, ONOS 1.3.0 SNAPSHOT, Mininet 2.2.1, Docker version 1.6.0. and OpenFlow 1.3.

3.1.5 Booting procedure

In order for the system to behave properly during the experiments, a series of steps must be taken to ensure proper configuration and stability, benefiting the analysis process and minimizing eventual noises in our results. First, the initialization starts by bringing up the Docker containers with images that have ONOS pre-installed. They require some time to boot up, since ONOS by default fetches its packages using Maven bundles (Apache Software Foundation, 2015a), installs and then configures those packages (we override that process by servicing packages locally, thus ensuring the exact same environment throughout our experiments). Second, the control instances need to be clustered together. After that process has finished, we can instantiate the Mininet network nodes (switches and hosts) and connect the control instances to them. In order to avoid an ARP storm – since, at first, the OVS switches are not connected with their controllers and run in L2-switching mode – we need to start them with STP enabled. Over time, the ONOS instances will recognize the topology. ONOS will not be able to recognize the disabled ports through its topology discovery mechanism at first as we started the switches with STP enabled, and thus not all links will be available.

Once the system has stabilized, we move on to disable STP for each OVS switch in our topology. As this process runs, we see that ONOS manages to recognize the previously inactive links. Next, we reassign the mastership roles among ONOS instances (according to the mapping mechanism provided by (MULLER et al., 2014) for the Abilene topology). Finally, we make sure that ONOS has recognized each one of the hosts in the network by sending ICMP echo requests among them utilizing the *ping* tool. This makes our testbed experiments focus on performance after the discovery process, aiming at evaluating the actual management process involving the control plane.

3.1.6 Assumptions

We based our experiments around the following assumptions and configurations:

- *The synchronization links between control instances are dedicated.* This means that synchronization traffic is not handled by the datapath elements – instead, we instantiate an individual, separate control network switch to handle traffic among control instances in a bridged configuration. When the instances are geographically separated, we apply delay to their links using traffic control according to datapath topology constraints. In real distributed control plane WAN scenarios, it is expected that the instances would instead take a share of the capacity of the data network to synchronize;
- *The network uses an in-band scheme for OpenFlow control messages.* Given the geographic scale of the scenarios we are considering, it seems reasonable use the in-band mode from OVS. Thus, there is not a complete control network dedicated to provide connectivity between control instances and forwarding devices. The cost of maintaining such separate network in a real production setting, with extra links and devices dispersed over large regions, would be prohibitively large;
- *The control instances run a reactive forwarding application.* The focus of the present study is on understanding how fast the control plane reacts to certain network events for each of the configurations under test. We would like to determine how the stress on the control plane would affect network-wide operations in each scenario defined. We leave the comparison between centralized and distributed control planes in proactive networking scenarios – such as network resource provisioning and allocation – for future work.

3.2 Topology scenarios

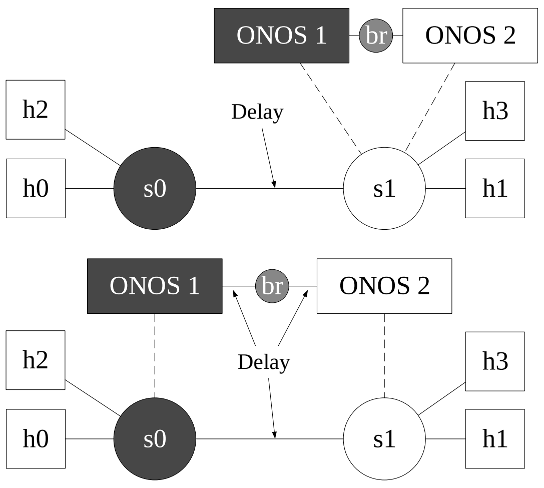
The topologies employed in our evaluation have been chosen according to how their characteristics would contribute to our analysis. We considered the ability to provide generalizable results as well as lead to findings that would be representative of real world scenarios. This way, we defined two types of toplogies for our evaluation, which are presented below.

3.2.1 Simple Linear Topology

First of all, we wanted to isolate particular behaviors from each kind of control organization under testing. Such requirement led us to devise a simpler, general topology, allowing us to identify and capture important features. Figure 3.2 illustrates the topology in question, referred to as *simple linear topology* throughout the document. In this topology, we limit the number of control instances, as well as the number of forwarding devices, to 2. Each forwarding device has two hosts connected to it. The idea behind these design choices was to mimic, with a simplified, generic scenario, two types of interactions: *local* interactions among co-located hosts (hosts connected to the same switch) and *global* interactions between geographically separated hosts (each host connected to a different switch). Two control instance distribution schemes have been employed: one in which both instances are connected to the same switch ($S1$, in this case) – centralized – and another where each instance would be located in a different region and connected to a different forwarding device – distributed.

For this scenario, we varied the propagation delay imposed in the link between the forwarding devices in order to better understand the effects of a WAN scenario on each control plane scheme under evaluation. In both control plane schemes evaluated, there is a synchronization channel connecting the forwarding devices (using an OVS bridge), but only in the decentralized case we apply delay to that channel. In the centralized case, we expect to measure lower latency for local interactions that happen closer to the control instances (e.g. $h1{-}h3$ – please refer to Figure 3.2), whereas both global interactions (e.g. $h0{-}h1$) and local interactions that occur far from it (e.g. $h0{-}h2$) should present worse results, but lower variability in general. As for the decentralized case, we expect to obtain highly variable latency results (mainly due to state synchronization issues), with high latency still. However, in the case of local interactions, given a proper controller-to-switch mapping, we expect to see low latency in interactions for both regions of the network, as each control instance would be able to simply configure a neighboring forwarding device without as much need for synchronization.

Figure 3.2 – Centralized (top) and distributed (bottom) cases for the simple linear evaluations. Notice that, in the centralized case, both control instances are located in the $S1$ region.



Source: by author.

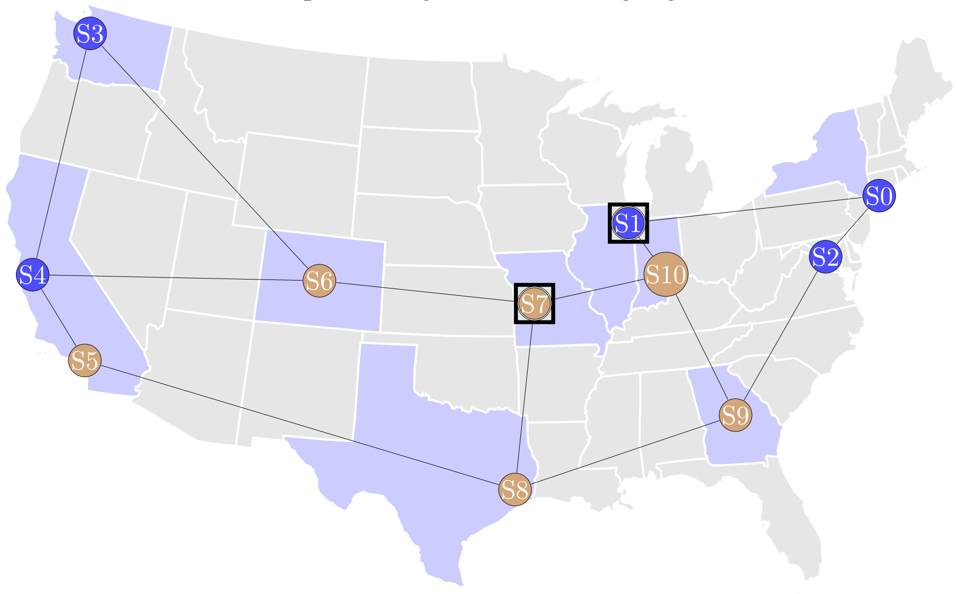
3.2.2 Abilene Topology

The second topology, Abilene, is closer to a real world WAN scenario. It is representative of current WAN deployments, mostly in terms of global scale. There, a distributed control plane deployment would be, presumably, recommended for enabling resilience and lower-latency management. The topology map, illustrated in Figure 3.3, is comprised of 11 nodes located in strategic positions throughout the United States territory. It is characterized also by having three internal loops (allowing path diversity among nodes) and a diameter of 5. We leveraged information obtained from (KNIGHT et al., 2011) to determine node connectivity and link latency for this topology.

By making use of the controller positioning and controller-to-switch mapping methodology from (MULLER et al., 2014), we can determine that the ideal number of controllers for this topology is 2 when using a distributed control plane scheme. These two controller instances are each located at $S1$ and $S7$ regions. The resulting mappings are also illustrated in the figure: the controller positioned at $S7$ is master of 6 forwarding devices in the topology ($S5$, $S6$, $S7$, $S8$, $S9$, $S10$), and the one located at $S1$ controls the remaining 5 ($S0$, $S1$, $S2$, $S3$, $S4$).

For the sake of simplicity, we compare this distributed scheme scenario with two centralized scenarios: one in which the two instances are located at $S7$ and another where both are positioned at $S1$ – the same locations used in the distributed case. This way, we intend to identify cases in which the separation has provided any kind of advantage over the centralized scheme. We could make attempts to analyze the results of centralized scenarios at various other locations of the network, not being restricted to $S1$ and $S7$. However, the whole process would take a long time to finish. With the current conditions, it takes about 22 hours to properly evaluate a given scheme in this topology – not to mention the laborious procedure of bootstrapping the testbed. Also, the process of making a more extensive analysis wouldn't provide us a much more significant contribution towards understanding the trade-offs among the different positioning schemes.

Figure 3.3 – Abilene network topology map. Color key indicates the control mastership mapping of forwarding devices. Control instance positioning locations are highlighted at $S1$ and $S7$.



Source: by author.

3.3 Experiments

In the following sections, we describe the methodology employed in our experiments to compare physically distributed and centralized control planes. They are executed considering an already bootstrapped emulation environment as precondition (as discussed in Section 3.1.5). Section 3.3.1 describes our metric on data plane latency involving control plane reactivity. Section 3.3.2, on the other hand, introduces our second metric, which focuses on determining the effects of the employed control plane configuration on intensive data plane interactions.

3.3.1 Round-Trip Time between hosts

Our firs metric is a measure of the Round-Trip Time between hosts of the topology under testing for each evaluated control plane organization. We use this evaluation metric both in the Abilene and the simple linear topology scenarios. With this metric – which is tightly associated with our first question – we are interested in assessing the total Round-Trip Time between each pair of hosts in the network when forcing the control instances to reactively interact with the forwarding elements to forward packets. By doing that, we expect to better understand the responsiveness of the underlying controller scheme whenever it is required for the control plane to act on datapath events.

For each pair of hosts, we perform 30 repetitions of ICMP echo requests using the *ping* tool. These requests require the forwarding devices between to interact with their controllers, in order to reactively set up a path that allows connectivity to be maintained among hosts. After each RTT interaction measured, we wait for 25 seconds. That is more than enough time for the control plane instances to clear the forwarding table entries related to the ICMP traffic in the forwarding devices (i.e. waiting until an OpenFlow *idle timeout* expires) and stabilize in the standard ONOS implementation. Given these conditions, we assume that each of the interactions is independent of the others.

In the first topology – the simple linear one, Figure 3.2 – we calculate the RTT between pairs of hosts with a varying delay parameter for the link between the forwarding devices. In the centralized case, both control instances are within the same location ($S1$), so there is no need for applying delay in their synchronization links. We do not attempt a different controller location for this scheme such as $S0$, since we would fall into a symmetric case. In the decentralized case, each instance is co-located with its corresponding forwarding device. We apply exact same delay as the one between the datapath switches to their synchronization channel. The controller-to-switch mapping is the same in every situation: instances in the distributed case are masters of their neighboring switches, whereas each instance in the centralized case controls either the distant switch ($S0$) or the local one ($S1$). We vary the delay parameter linearly between $0ms$ and $20ms$, with steps of $5ms$ in between (the $0ms$ value corresponds to an ideal case, where no propagation delay would influence the testbed). In order to facilitate the evaluation and analysis processes, we chose to limit ourselves to the following representative pairs of interactions (refer to Figure 3.2 for clarification): $h0{-}h1$ (global interaction), $h0{-}h2$ and $h1{-}h3$ (local interactions, each at a different region of the network).

As for the Abilene topology, we calculate the RTT for every combination of hosts in the network. The network link propagation delays were pre-computed according to the properties of the actual network. Since there is one host connected to each forwarding devices of the topology – 11 hosts in total – there are 55 pairings, or 110 possible interactions in total (i.e. considering $(src : h0, dst : h1)$ and $(src : h1, dst : h0)$ as two separate interaction cases). In the decentralized control plane evaluation for this topology, we consider the delay between control instances to be sum of the propagation delays of the shortest path between $S1$ and $S7$, as would be the case in a real world scenario where the synchronization traffic shares link resources with the data plane elements.

3.3.2 Host throughput

The second metric is a measurement of the throughput among hosts in the data plane when the control plane is required to reactively act on their interactions. The main goal of this evaluation is to obtain a notion of how the host throughput is affected for each type of control plane scheme. We once again make use of different values of a delay parameter that represents WAN delay propagation effects. We use this metric in conjunction with the simple linear topology only (Figure 3.2). This provides a simple, but clear way of investigating the answer to the second question we posed, related to measuring the effects of the control plane distribution on regular network operation. For further guidance in that effort, we also measure the overall control traffic throughput in both forwarding devices and in the controller synchronization channel.

In the experiment that uses this metric, we limit our evaluation to global interaction throughput between $h0$ and $h1$ – both in the outgoing and incoming directions for each. We generate traffic originating from both of the hosts, and measure that traffic for a window of 100 seconds. The hosts run each a script that generates UDP packets targeted at the IP address of one another. The traffic that is generated produces packets scheduled according to a Poisson process with mean interval of $10ms$. The sizes of the packets vary exponentially around a mean of 1024 bytes, upper-bounded by the maximum Ethernet payload size of 1500 bytes. The seeds used in the traffic generation are different for each host, but the same in every scenario evaluated. Also, we perform IP spoofing in the source address of the packets, so as to force the controller to constantly interact with flow setups. This experiment is built to simulate a highly demanding situation, where multiple events need to be handled by the control plane in a short period of time. There is no need for a socket handler for incoming UDP packets in either host, as we compute those results directly on host interfaces using *tcpdump*. Similarly to the first metric with the same topology, we vary the link delay parameter for values from $5m$ to $20ms$ of propagation delay, with steps of $5m$ (we don't consider the ideal case of no delay for this host throughput metric).

4 RESULTS

In this chapter, we present the results obtained in the SDN WAN experiments introduced previously. As stated in Chapter 3, we aim at answering the following questions: (i) *how does the inherent propagation latency of WAN networks affect general communication in a distributed control plane setup*? and (ii) *how does the control plane scheme adopted impact on intensive data plane communication*? Towards this end, Section 4.1 addresses the experiments using the first metric (round-trip time between hosts), which attempts to answer our first question, related to control plane responsiveness in reaction to network events. These results have been obtained both for the Abilene and simple linear topologies. Section 4.2, in turn, addresses the second experiment metric (host throughput), seeking to understand the effects of control plane behavior under intensive load on network traffic.

4.1 Control Plane Responsiveness

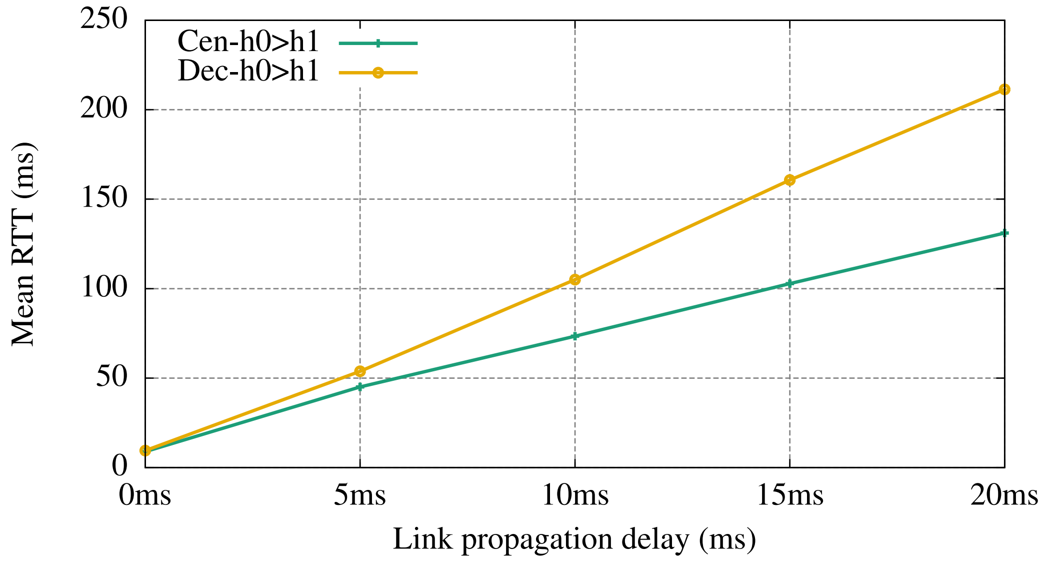
This section presents our findings for the first metric of our study. It is separated according to each the topology employed in our analysis: Section 4.1.1 attempts to identify general results and principles by analyzing the experimental results over the simple linear topology. Section 4.1.2 moves this discussion towards real world WAN deployments, providing results on the performance of distributed and centralized globally deployed control planes for these scenarios.

4.1.1 Simple Linear Topology

In the simple linear topology (shown in Figure 3.2), we want (i) to understand how latency over a single hop can affect communication between hosts, considering a distributed control plane (where latency impacts the synchronization of control instances); and (ii) to isolate and analyze the forwarding behavior of both local and global traffic under each type of controller scheme. We show the results in Figures 4.1 and 4.2. More specifically, Figure 4.1 illustrates our findings for the RTT for global interactions in the topology, whereas Figure 4.2 focuses on the the local interactions. These figures present the mean RTT values between each pair of hosts, in both directions, for different values of a varying link delay parameter.

As expected, Figure 4.1 indicates that the effective data plane latency for global interactions, involving hosts in different regions of the network, grows linearly with the increase in delay between those regions. This happens with both centralized and decentralized control plane schemes. The result also indicates that the disparity between latencies in the two schemes becomes more significant as the link delay increases, favoring the centralized one. In the case of $20ms$ of link delay, the increase of latency from the centralized to the decentralized case is of 61%. Even though this is a result for an extreme case (very large single link delay), it helps us to illustrate the geographical scalability effects of each scheme under evaluation.

Figure 4.1 – Mean latency with a varying link delay parameter for *global* interactions in each control plane scheme (Simple Linear Topology).

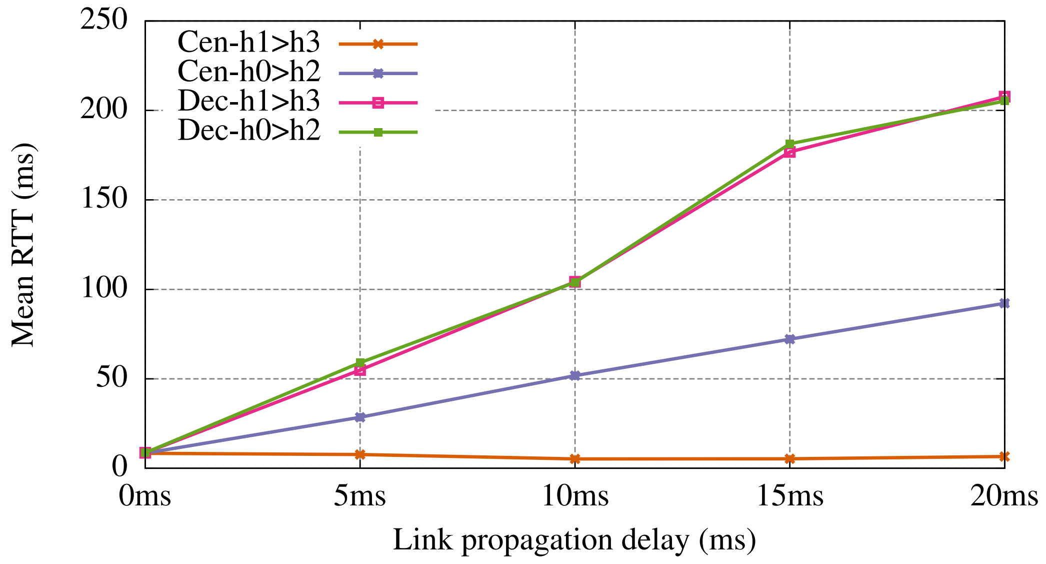


Source: by author.

To gain a better understanding, we look at the interactions for the centralized case for this point. First, host $h0$ sends an ICMP echo request to forwarding device $S0$. Since it does not have a rule in its forwarding table matching the incoming packet, it sends a *packet-in* message to the network controller (located at the $S1$ region) using the inter-region link. After about $20ms$, this packet arrives at $S1$, which in turn sends it to the controller. The controller computes the forwarding state update that needs to be performed at $S0$ in order to allow similar packets to flow in the same direction. Then, it sends both a *flow-mod* and a *packet-out* OpenFlow messages back to $S0$ ($20ms$ more). Once $S0$ receives those messages, it immediately installs a forwarding rule and sends the packet to $S1$. $S1$ then performs the same interaction as the one that happened between $S0$ and the controller (recall that the control plane runs a *reactive forwarding* application), but this time not being limited by link delays. An analogous process occurs for the corresponding ICMP echo reply, resulting in at least $120ms$ of additive delay due to the inter-region link. Thus, the latency measured at the $20ms$ delay point for the centralized case ($131.06ms$) is consistent with this setting (without considering the processing overhead).

This is in contrast with the decentralized case. Should the control instances be operating completely independently of each other, the total response time would potentially drop to around $40ms$ (without processing overhead), as each forwarding device is co-located with its respective controller. However, the coordination among the control instances using their synchronization channel proved to be the major drawback of this approach, bringing the mean perceived latency to $211.36ms$. There were a few cases in which this measured latency was in the order of $175.0ms$, indicating a variability that depends on the synchronization state in which the instances are when the requests arrive at them. These observations suggest that ONOS employs strong consistency in network updates.

Figure 4.2 – Mean latency with a varying link delay parameter for *local* interactions in each control plane scheme (Simple Linear Topology).



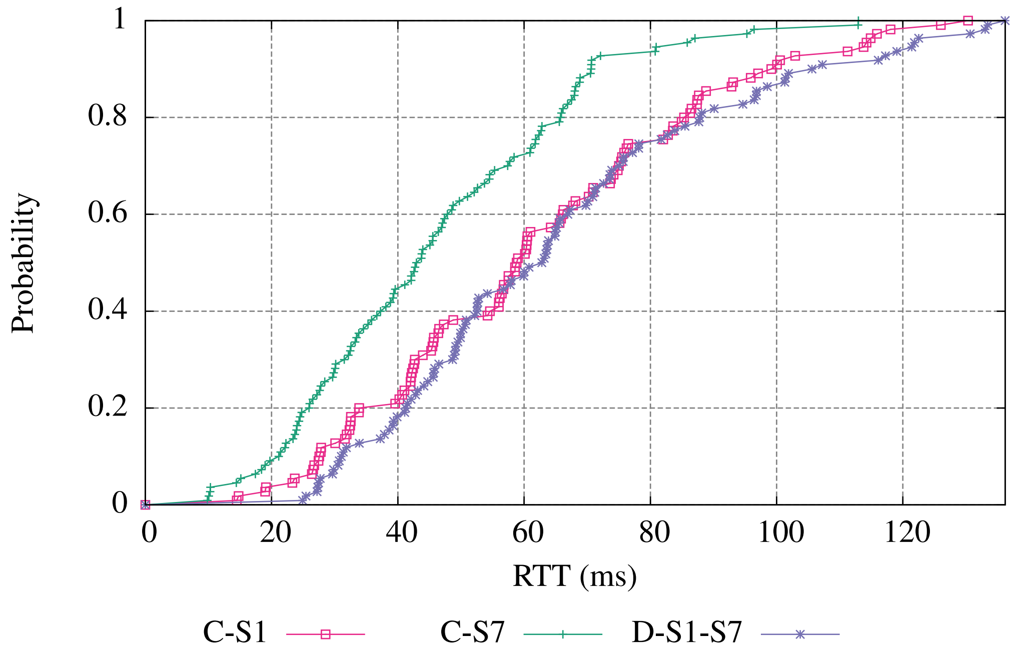
Source: by author.

Next, Figure 4.2 presents the results for local interaction latencies. We can see that the two local interactions involving the decentralized control plane yielded approximately the same response times, while interactions in the centralized case presented different response times. This happened because communication between $h1-h3$ did not involve the inter-region link, in contrast to communication between $h0-h2$ and all communication of the distributed case. Notice that the measured latency for the distant RTT interaction (Cen-h0>h2) was smaller in this case when compared to the global interaction from the previously presented results (Cen-h0>h1), at about $92.2ms$. That happens because the sequence of events to complete the RTT interaction in this case involves less utilization of the inter-region link. In the decentralized case, the synchronization overhead was quite large like in the global interaction results from before. This indicates that, even for events that could be easily handled by a single controller instance, they still have to coordinate before taking action.

4.1.2 Abilene Topology

With this experiment, we are interested in understanding how some of the effects observed in the simplified case study with the simple linear topology manifest themselves in an actual WAN deployment scenario with the Abilene topology (Figure 3.3). In this topology, we do not find propagation delays as large as those used in the previous experiment, being limited to around $1.71ms$ in the longest link. However, due to the larger number of forwarding devices, along with the presence of more links, even more interactivity takes place between the control and data planes of the network. We compare three positioning cases for this topology: one distributed case in which the two deployed instances are positioned each at $S1$ and $S7$ regions, and two centralized cases where both instances are either at $S1$ or $S7$.

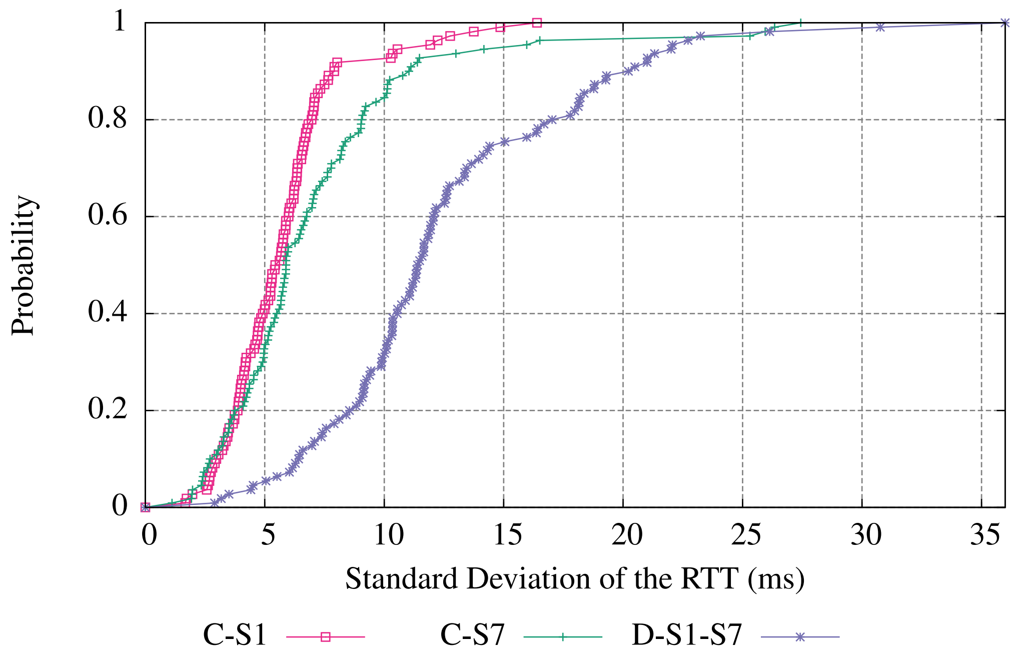
Figure 4.3 – CDF of mean interaction latencies for each scenario.



Source: by author.

Since the number of interactions evaluated is quite larger for this network, we leverage the use of CDFs to display these results. Figure 4.3 presents the CDF for the mean RTT values measured among hosts in this topology. It is noticeable that having the control plane centrally clustered around $S7$ produced, in general, lower mean latency results than the other schemes. The highest mean value obtained in this case stands at $112.5ms$, whereas the ones obtained with a centralized scheme at $S1$ and distributed among $S1$ and $S7$ topped at $130.33ms$ and $136.20ms$ respectively. We recognize, however, that the decentralized and centralized at $S1$ cases present quite similar results for this visualization. Also note that, with this plot, we are comparing all mean results in general, without more specific information for analyzing individual host pairing interactions.

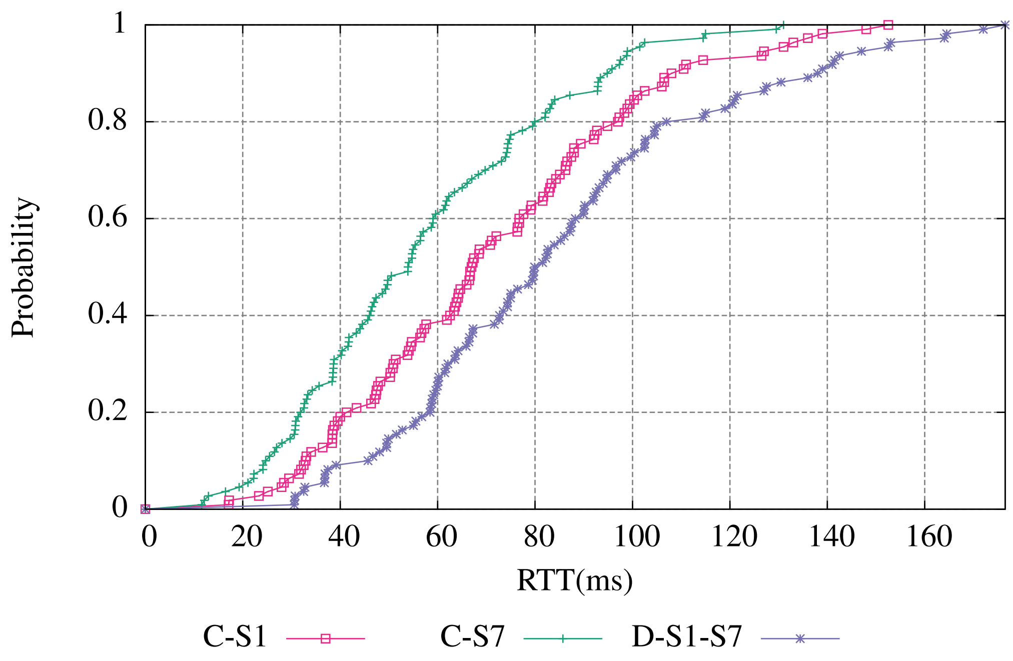
Figure 4.4 – CDF of standard deviation of interaction latencies for each scenario.



Source: by author.

Next, we analyze the variability of RTTs measured on each scheme. Figure 4.4 illustrates the standard deviation CDF for the same experiments (recall that each interaction is repeated 30 times). This time, we see that having the control instances centralized at $S1$ provides less variability in general – thus, it also tends to be more *predictable* than the other two schemes for most interactions in the data plane. While $S7$ does not fare very far behind in up to 90% of the cases when compared to $S1$ (in more than 80% of the interactions their standard deviation is below $10ms$), we can see that the variability present in the decentralized case is amplified, since control instances need to coordinate between themselves. Recall from the previous experiments with the Simple Linear Topology that increasing delays in the synchronization channel can greatly diminish overall data plane latency when considering the control plane overhead in the interactions.

Figure 4.5 – CDF of the 95th percentile of interaction latencies for each scenario.



Source: by author.

Taking individual pairs of hosts and comparing the mean RTT between all evaluated schemes, we find that the scheme where the control instances are located at $S7$ provides the lowest mean latency among the three for about 76% of the possible interactions. There was no pair for which the decentralized scheme performed better than the centralized ones. For the rest of the interactions (24%), $S1$ provided better mean latency than the other two. By analyzing the results closely, we see that the positioning scheme in which we centralize the control instances at $S1$ does better than the others mostly in cases where either the origin/destination node is much closer to the $S1$ forwarding device when compared to $S7$ (e.g. from $h0$, $h1$ and $h10$). Since $S7$ is located at a central region of the topology and connects to more forwarding devices, it benefits from that in terms of the overall latency to all other forwarding devices when compared to $S1$, which is located at the periphery of the network.

Moreover, Figure 4.5 illustrates the worst-case scenario latencies measured throughout the experiments – taking only the $95th$ percentile of the sample from each interaction, in order to discard outliers. This result emphasizes the differences between the three control plane placement schemes, showing that the decentralized control plane provides the worst performance of the three for the RTT metric for most situations. The highest $95th$ percentile latencies measured for each of the schemes were $152.5ms$, $131.0ms$ and $176.5ms$ for the $S1$ and $S7$ centralized schemes and the decentralized one, respectively.

4.2 Effects of Intensive Data Plane Interactions

This section presents the final results for our control plane distribution study. Given the nature of the metrics employed in this experiment, we decided to explore them only on the simple linear topology (Figure 3.2), as it enables us to better visualize the underlying network behavior. We believe this type of isolated behavior would be more difficult to extract if we employ the exact same methodology on topologies such as Abilene (Figure 3.3), so we leave this analysis to future work. Section 4.2.1 presents our results for the second experiment, and also concludes this chapter.

4.2.1 Simple Linear Topology

In this experiment, we analyze the impact of control distribution in the performance perceived by the data plane elements. In the case of this analysis, we limit our evaluation to global interactions occurring between the $h0$ and $h1$ hosts in the simple linear topology. The link delay parameter is varied between $5ms$ and $20ms$, with steps of $5ms$ in between. Hence, we are focusing on how each control plane scheme behaves in face of the decreasing network visibility caused by the increased delay among regions and how each control scheme is affected by it.

Summarizing the experiment overview from the previous chapter, we generate UDP traffic in both $h0$ and $h1$, and make them target each other using spoofed source IP addresses. This forces the forwarding devices to constantly perform *packet-in* requests to their respective control instances in order to forward the packets towards the hosts. Should the control plane not respond in time due to network delay and control mechanism constraints, we will observe a drop in the ratio between data that was originally sent and what was effectively received in the destination host. The duration of this experiment is $100s$ for each scenario combination (control plane organization and link delay).

Table 4.1 – Resulting mean aggregate traffic loads measured in forwarding devices, synchronization bridge and network hosts for varying sizes of the propagation delay of the inter-region link (values in kbps).

Scheme	**Entity**	**5ms**	**10ms**	**15ms**	**20ms**
Centralized	$S0$	395.24	396.88	397.91	397.74
	$S1$	1166.01	1171.03	1169.83	1168.29
	$synch$	1277.88	2269.18	1716.17	1716.99
	$aggrTX$	78.23	78.60	78.34	78.65
	$aggrRX$	77.89	78.41	78.29	78.59
Distributed	$S0$	716.68	707.49	710.88	606.12
	$S1$	717.38	709.50	709.06	694.24
	$synch$	1075.81	1003.64	954.93	781.95
	$aggrTX$	78.46	78.43	78.17	77.60
	$aggrRX$	78.33	78.31	77.87	72.14

Source: by author.

Table 4.1 presents the aggregate results for the experiment. For each control plane scheme analyzed, we display the aggregate mean throughput measured in each element of the network. In the case of forwarding devices, we measure this throughput by collecting their internal forwarding table counters for control messages (considering both OpenFlow messages relayed due to in-band control and direct exchanges with the control instances). As for synchronization messages among control instances, we measure the counters directly from the synchronization bridge – $synch$ rows. We monitor the network interfaces of both hosts $h0$ and $h1$ using *tcpdump* to filter the UDP packets involved in the interactions. That is necessary as the controller constantly floods the network with LLDP packets for network discovery. We keep statistics on sent and received bytes of UDP packets by hosts. Also, we sum values of the same type to simplify their presentation (e.g.: $h0\,TX + h1\,TX = aggrTX$ in the results table).

The results show that the throughput observed at the aggregate host traffic in the centralized scheme presented little change for varying link delay values. In the decentralized case, the hosts presented a slight degradation in effective aggregate throughput for the $15ms$ and $20ms$ cases. Even though the traffic rates between hosts were not aggressive, the fact that each packet sent has a different source IP address required the control plane to constantly interact with the data plane throughout the exchange process. This behavior led to packet loss across all tested scenarios. Interestingly, packets sent specifically from $h0$ to $h1$ in the centralized scenario presented no misses in delivery at all for every value of the delay parameter, in contrast with traffic going the opposite way. Conversely, in the decentralized case with the delay parameter set to $20ms$, we measured data losses of 12.74% in bytes for interactions starting at $h1$ to $h0$.

The results for the centralized control plane synchronization traffic indicate higher rates of this kind of traffic than those from the decentralized case. We can see from the results that higher delays in the synchronization channel not only imply in delayed network view, but can also impact in the available throughput in that channel for the decentralized case. On the other hand, notice the disparity of load in the switches for each of the schemes evaluated. In the centralized scheme, the average load on the $S1$ forwarding device was way higher than that of $S0$ (almost 3 times as large), as all OpenFlow traffic targeting $S0$ needs to be forwarded through it. This behavior happens even though the traffic being originated in either region is of the same nature. Meanwhile, the control traffic is balanced evenly among forwarding devices in the decentralized case.

5 CLOSING REMARKS AND FUTURE WORK

This work presented a novel study which focuses on understanding the effects of control plane distribution in WAN settings. Our contributions to this area are, among others: (i) we introduce the motivation for the problem of control plane distribution effects in WAN networks; (ii) we develop a new testbed configuration, using production-quality software components, aiming for emulation of the effects of globally deployed networks; and (iii) we provide results that allow for a comparison between centralized and distributed control plane schemes in WAN networks.

Our experimental results indicate that, despite the potential benefits of globally distributed control planes in terms of resilience, their responsiveness to network events is not necessarily better than that of a centralized approach. The increased latency between control instances incurs in effects that range from larger and more variable latency from the control plane to network events to diminished data plane throughput. These effects are caused mainly due to the constant need for synchronization among instances, and puts into question the feasibility of reactive distributed control planes with strong network state consitency constraints.

We also identified that distributed control planes allow for the control communication over the network infrastructure to be more balanced among forwarding devices when using in-band control. This is in contrast with a centralized control plane, which behaves as a focal point for all control traffic generated in the network. Still, the need for a synchronization channel to be maintained among the globally separated instances poses a challenge to the performance and reliability of distributed control planes.

Although this study provides the initial insights on globally-deployed control plane distribution trade-offs, there are plenty of open research fronts that can explored in this area. For instance, we have not devised a comparison between centralized and distributed control planes in proactive control approaches, such as in those involving network resource provisioning. Also, it was identified during this study that the current Open vSwitch in-band scheme implementation does not encompass multipath alternatives for guaranteed delivery of control packets, greatly harming the deployability of resilient distributed control planes over it. Finally, we identified research potential in analyzing the trade-offs discussed in this study in other types of globally deployed topologies. We leave these issues and open research questions to be explored as future work.

REFERENCES

A. R. Curtis et al. DevoFlow: scaling flow management for high-performance networks. *ACM SIGCOMM CCR*, New York, NY, USA, v. 41, n. 4, 2011. ISSN 0146-4833.

AKELLA, A.; KRISHNAMURTHY, A. A Highly Available Software Defined Fabric. In: **Proceedings of the 13th ACM Workshop on Hot Topics in Networks**. New York, NY, USA: ACM, 2014. (HotNets-XIII), p. 21:1–21:7. ISBN 978-1-4503-3256-9. Available from Internet: <http://doi.acm.org/10.1145/2670518.2673884>.

Apache Software Foundation. **Apache Maven Project**. 2015. Available from Internet: <https://maven.apache.org/>.

Apache Software Foundation. **Karaf 4.0.3**. 2015. Available from Internet: <http://karaf.apache.org/>.

CAI, Z.; COX, A. L.; NG, T. E. Maestro: A system for scalable openflow control. **Structure**, 2010.

CANINI, M. et al. A Distributed and Robust SDN Control Plane for Transactional Network Updates. In: **INFOCOM**. [S.l.: s.n.], 2015.

CASADO, M. et al. Fabric: a retrospective on evolving SDN. In: **HotSDN**. New York, NY, USA: ACM, 2012. p. 85–90. ISBN 978-1-4503-1477-0.

DAI, W. et al. R-sdn: A recusive approach for scaling sdn. In: **Information and Communications Technologies (ICT 2014), 2014 International Conference on**. [S.l.: s.n.], 2014. p. 1–6.

Docker, Inc. **Docker - Build, Ship and Run Any App, Anywhere**. 2015. Available from Internet: <https://www.docker.com/>.

ERICKSON, D. The beacon openflow controller. In: ACM. **Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking**. [S.l.], 2013. p. 13–18.

GEROLA, M. et al. Icona: Inter cluster onos network application. In: **Network Softwarization (NetSoft), 2015 1st IEEE Conference on**. [S.l.: s.n.], 2015. p. 1–2.

GEROLA, M. et al. ICONA: inter cluster ONOS network application. **CoRR**, abs/1503.07798, 2015. Available from Internet: <http://arxiv.org/abs/1503.07798>.

Gregory Gee. **Running Open vSwitch in Network Namespace**. 2014. Available from Internet: <http://techandtrains.com/2014/02/08/running-open-vswitch-in-network-namespace/>.

GUDE, N. et al. NOX: towards an operating system for networks. **ACM SIGCOMM Computer Communication Review**, ACM, v. 38, n. 3, p. 105–110, 2008.

HELLER, B.; SHERWOOD, R.; MCKEOWN, N. The controller placement problem. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 42, n. 4, sep. 2012. ISSN 0146-4833.

HONG, C.-Y. et al. Achieving high utilization with software-driven wan. In: ACM. **ACM SIGCOMM Computer Communication Review**. [S.l.], 2013. v. 43, n. 4, p. 15–26.

HUNT, P. et al. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In: **USENIX Annual Technical Conference**. [S.l.: s.n.], 2010. v. 8, p. 9.

KNIGHT, S. et al. The internet topology zoo. **Selected Areas in Communications, IEEE Journal on**, v. 29, n. 9, p. 1765–1775, October 2011. ISSN 0733-8716.

Koponen, T. et al. Onix: a distributed control platform for large-scale production networks. In: **USENIX OSDI**. [S.l.: s.n.], 2010.

KREUTZ, D. et al. Software-Defined Networking: A Comprehensive Survey. **Proceedings of the IEEE**, v. 103, n. 1, Jan 2015.

KUMAR, A. et al. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In: **Sigcomm '15**. [S.l.: s.n.], 2015.

KWAK, M. et al. Raon: A recursive abstraction of sdn control-plane for large-scale production networks. In: **Information Networking (ICOIN), 2015 International Conference on**. [S.l.: s.n.], 2015. p. 426–427.

LAKSHMAN, A.; MALIK, P. Cassandra: a decentralized structured storage system. **ACM SIGOPS Operating Systems Review**, ACM, v. 44, n. 2, p. 35–40, 2010.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: ACM. **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**. [S.l.], 2010. p. 19.

LEVIN, D. et al. Logically centralized?: state distribution trade-offs in software defined networks. In: **HotSDN**. [S.l.]: ACM, 2012. ISBN 978-1-4503-1477-0.

Linux Foundation. **Netem**. 2015. Available from Internet: <http://www.linuxfoundation.org/collaborate/workgroups/networking/netem>.

MATNI, N.; TANG, A.; DOYLE, J. C. A case study in network architecture tradeoffs. In: **Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research**. New York, NY, USA: ACM, 2015. (SOSR '15), p. 18:1–18:7. ISBN 978-1-4503-3451-8. Available from Internet: <http://doi.acm.org/10.1145/2774993.2775011>.

MCCAULEY, J. **POX: A Python-based OpenFlow Controller. http://www.noxrepo.org/pox/about-pox/**. 2014. Available from Internet: <http://www.noxrepo.org/pox/about-pox/>.

MCCAULEY, J. et al. **Recursive SDN for Carrier Networks**. [S.l.], 2015.

MEDVED, J. et al. Opendaylight: Towards a model-driven sdn controller architecture. In: **World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a**. [S.l.: s.n.], 2014. p. 1–6.

MULLER, L. et al. Survivor: an Enhanced Controller Placement Strategy for Improving SDN Survivability. In: **GLOBECOM**. [S.l.: s.n.], 2014.

N. McKeown et al. OpenFlow: enabling innovation in campus networks. **SIGCOMM Comput. Comm. Rev.**, v. 38, n. 2, 2008.

ON.LAB. **ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out.** 2015. Available from Internet: <http://onosproject.org/>.

Open vSwitch. **Open vSwitch: Production Quality, Multilayer Open Virtual Switch**. 2013. Available from Internet: <http://openvswitch.org/>.

P. Berde et al. ONOS: Towards an Open, Distributed SDN OS. In: **HotSDN**. [S.l.]: ACM, 2014. ISBN 978-1-4503-2989-7.

PANDA, A. et al. CAP for networks. In: **HotSDN**. [S.l.]: ACM, 2013.

PFAFF, B. et al. The design and implementation of open vswitch. In: **12th USENIX Symposium on Networked Systems Design and Implementation**. [S.l.: s.n.], 2015.

PHEMIUS, K.; BOUET, M.; LEGUAY, J. Disco: Distributed multi-domain sdn controllers. In: **Network Operations and Management Symposium (NOMS), 2014 IEEE**. [S.l.: s.n.], 2014. p. 1–4.

REITBLATT, M. et al. Abstractions for Network Update. In: **ACM SIGCOMM 2012**. New York, NY, USA: ACM, 2012. (SIGCOMM '12), p. 323–334. ISBN 978-1-4503-1419-0. Available from Internet: <http://doi.acm.org/10.1145/2342356.2342427>.

Ryu SDN Framework Community. **RYU network operating system.** 2015. Available from Internet: <http://osrg.github.com/ryu/>.

S. Jain et al. B4: experience with a globally-deployed software defined wan. In: **ACM SIGCOMM**. [S.l.: s.n.], 2013. ISBN 978-1-4503-2056-6.

SCHMID, S.; SUOMELA, J. Exploiting Locality in Distributed SDN Control. In: **Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking**. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 121–126. ISBN 978-1-4503-2178-5. Available from Internet: <http://doi.acm.org/10.1145/2491185.2491198>.

The Internet2 Community. **Advanced Networking | Internet2**. 2015. Available from Internet: <http://www.internet2.edu/products-services/advanced-networking/>.

TOOTOONCHIAN, A.; GANJALI, Y. HyperFlow: A Distributed Control Plane for OpenFlow. In: **USENIX INM/WREN**. [S.l.: s.n.], 2010.

TOOTOONCHIAN, A. et al. On Controller Performance in Software-defined Networks. In: **Hot-ICE**. [S.l.]: USENIX, 2012.

YAN, J.; JIN, D. Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation. In: ACM. **Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research**. [S.l.], 2015. p. 27.

YEGANEH, S. H.; GANJALI, Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In: **ACM HotSDN**. [S.l.: s.n.], 2012. ISBN 978-1-4503-1477-0.

YEGANEH, S. H.; GANJALI, Y. Beehive: Towards a Simple Abstraction for Scalable Software-Defined Networking. In: **Proceedings of the 13th ACM Workshop on Hot Topics in Networks**. New York, NY, USA: ACM, 2014. (HotNets-XIII), p. 13:1–13:7. ISBN 978-1-4503-3256-9. Available from Internet: <http://doi.acm.org/10.1145/2670518.2673864>.

YU, M. et al. Scalable Flow-based Networking with DIFANE. In: **ACM SIGCOMM**. [S.l.: s.n.], 2010.

ZHANG, Y.; BEHESHTI, N.; TATIPAMULA, M. On Resilience of Split-Architecture Networks. In: **GLOBECOM**. [S.l.: s.n.], 2011. p. 1–6. ISSN 1930-529X.