UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

TOBIAS BRIGNOL PETRY

Avoiding Control Plane Partition in Software Defined Networks through Cellular Networks: assessing opportunities and limitations

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Marinho P. Barcellos

Porto Alegre
August 2015

CIP — CATALOGING-IN-PUBLICATION

Petry, Tobias Brignol

Avoiding Control Plane Partition in Software Defined Networks through Cellular Networks: assessing opportunities and limitations / Tobias Brignol Petry. – Porto Alegre: PPGC da UFRGS, 2015.

75 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2015. Advisor: Marinho P. Barcellos.

1. Software Defined Networks. 2. Cellular Networks. 3. OpenFlow. 4. Control Plane. 5. Partition. 6. Resilience. I. Barcellos, Marinho P.. II. Título.

"But seek ye first the kingdom of God, and his righteousness;
and all these things shall be added unto you."

— JESUS OF NAZARETH (MATTHEW 6:33 KJV)

ACKNOWLEDGMENTS

To God, Divine Master. To Him, all Glory, Honor and Praise. To Our Lady of Lourdes for the particular spiritual help in the most difficult moments of this course.

To my parents, Jairo and Ligia. He, as a close companion and witness to every effort put into this work. She, as a proud intercessor from eternity. To my brothers Jesus and Giovani Petry, for setting an example with successful academic and professional careers. To Marcos Davi, best friend and a teacher by nature, greatly missed. To Ana Carolina, the girlfriend who endured with patience my moments of anguish and understood my need to devote time to this work. To each and every friend not mentioned in particular, who shared the grief and joy.

To my advisor, Marinho Barcellos, who constantly pushed me to excellence and taught about critical thought and scientific writing. To all the teachers and the staff at the Institute of Informatics of UFRGS, some of whom I've been daily living with for the past ten years since I was an undergraduate. To my colleagues and friends at the Networking group, who gave me invaluable support and advice throughout the course. In particular, to my friend Rafael da Fonte, who co-authored a paper on the topic of this dissertation. To the Brazilian Catholic Chaplaincy Our Lady of Aparecida, for aiding my stay in London to present our paper.

AGRADECIMENTOS

A Deus, Divino Mestre. A Ele a Glória, a Honra e o Louvor. A Nossa Senhora de Lourdes pelo auxílio espiritual nos momentos mais difíceis do curso.

Aos meus pais, Jairo e Ligia. Ele acompanhou de perto todo o esforço investido neste trabalho. Ela, da eternidade, certamente intercedeu e está orgulhosa. Aos meus irmãos Jesus e Giovani Petry, por terem dado o exemplo em suas carreiras acadêmicas e profissionais. Ao Marcos Davi, saudoso melhor amigo e professor nato. A Ana Carolina, a namorada que pacientemente aguentou as angústias e entendeu o tempo que dediquei a este trabalho. A cada amigo não mencionado, que de alguma maneira partilhou das tristezas e alegrias.

Ao meu orientador, Marinho Barcellos, que constantemente me incentivou à excelência, e ensinou muito sobre pensamento crítico e escrita científica. A cada professor e colaborador do Instituto de Informática da UFRGS, com alguns dos quais tenho convivido a dez anos, desde o tempo da graduação. Aos meus colegas e amigos do grupo de Redes, que me deram grande apoio e conselhos valiosos. Em particular, ao amigo Rafael da Fonte, co-autor de artigo no tema da dissertação. À Capelania Católica Brasileira Nossa Senhora Aparecida, que me recebeu em Londres para que eu apresentasse nosso artigo.

ABSTRACT

Software Defined Networks simplify network programmability by detaching the control plane from forwarding devices and deploying it into a logically centralized controller. While this allows a clearer separation of concerns, it also creates a dependency between them. Failures in the control plane break the controller view of the network state and could render the network unusable if forwarding devices cannot be reached. The relevance of this problem has led to a range of proposals, including physical distribution of controller instances and delegation of concerns to forwarding devices. This dissertation features the proposal and evaluation of an architecture that leverages cellular data networks (4G) as control plane backup links. No previous work has explored this idea, despite the recent research intersecting SDN and wireless networks. The experimental evaluation provides insights towards answering three research questions: (i) *How is the behavior of control plane traffic affected by the characteristics of cellular links*, (ii) *how quickly is the control plane handed over to the backup link when a failure occurs* and (iii) *how well do network functions that rely on a snapshot of the network state behave on such an architecture*. Despite the expected higher latency of cellular links, this architecture maintains partial functionality of tasks that depend on global network awareness when failures occur in primary control links in a simple, affordable fashion. The degree to which the functionality of these tasks is maintained is directly related to its dependency on the timeliness of control plane reaction to network events. The main benefit of preventing control plane partition is to maintain a consistent global view of the network.

Keywords: Software Defined Networks. Cellular Networks. OpenFlow. Control Plane. Partition. Resilience.

Evitando a Partição do Plano de Controle em Redes Definidas por Software Através de Redes Celulares: avaliando oportunidades e limitações

RESUMO

Redes Definidas por Software ajudam a simplificar a programabilidade da rede ao desacoplar o plano de controle dos dispositivos de encaminhamento, e implementá-lo em um controlador logicamente centralizado. Apesar de permitir uma separação de conceitos mais clara, essa característica cria também uma relação de dependência entre controlador e dispositivos. Falhas no plano de controle prejudicam a visibilidade do estado da rede no controlador e podem tornar a rede inutilizável caso os dispositivos de encaminhamento sejam isolados. A relevância deste problema motivou uma série de propostas, incluindo a distribuição física de instâncias de controle e a delegação de tarefas aos dispositivos de encaminhamento. Esta dissertação contém a proposta e a avaliação de uma arquitetura que usa redes celulares de dados (4G) como enlaces reservas para o plano de controle. Nenhum trabalho anterior explorou esta ideia, apesar da pesquisa recente envolvendo Redes Definidas por Software e redes sem fio. A avaliação experimental permite uma melhor compreensão ao responder três perguntas: (i) *Como o comportamento do tráfego do plano de controle é afetado pelas características de enlaces celulares*, (ii) *quão rapidamente o plano de controle é migrado para o enlace reserva quando uma falha ocorre* e (iii) *como funções de rede que dependem do estado da rede em um instante se comportam em tal arquitetura*. Apesar da já esperada maior latência dos enlaces celulares, esta arquitetura mantém o funcionamento parcial de tarefas que dependem de visão global da rede quando falhas ocorrem nos enlaces primários, de maneira simples e com custo acessível. O grau de manutenção de tais tarefas é diretamente relacionado com sua dependência da rapidez de reação do plano de controle a eventos de rede. O principal benefício de prevenir a partição do plano de controle é a manutenção de uma visão global consistente da rede.

Palavras-chave: Redes Definidas por Software, Redes Celulares, OpenFlow, Plano de Controle, Partição, Resiliência.

LIST OF ABBREVIATIONS AND ACRONYMS

4G	Fourth generation (of mobile telecommunications technology)
DHT	Distributed Hash Table
FCS	Frame Check Sequence
IDS	Intrusion Detection System
IP	Internet Protocol
LAN	Local Area Network
LLDP	Link Layer Discovery Protocol
LTE	Long-Term Evolution (of 4G)
MAC	Media Access Control
MPTCP	Multipath TCP
NAT	Network Address Translation
NIC	Network Interface Controller (or Card)
OF	OpenFlow
OVS	Open vSwitch
PEP	Performance Enhancing Proxy
RFC	Request For Comments
RTT	Round-Trip Time
SDN	Software Defined Networks
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual LAN
WAN	Wide Area Network

LIST OF FIGURES

Figure 3.1 Cellular Links in SDN for shared data and control plane traffic 19
Figure 3.2 Cellular Links in SDN for east/westbound control traffic 20
Figure 3.3 Cellular Links in SDN for southbound control traffic 21
Figure 3.4 Basic experimental environment 23

Figure 4.1 Terminology for message exchanges in the RTT experiment................................ 25
Figure 4.2 Environment and metrics for the message exchange RTT experiments 27
Figure 4.3 Average RTT for both media in each scenario 33

Figure 5.1 Environment for the handover experiment 37
Figure 5.2 Handover procedure description... 39
Figure 5.3 OpenFlow connection handshake on failover 40
Figure 5.4 Multipath TCP control channel 42

Figure 6.1 IP prefix binary tree 45
Figure 6.2 Environment for experiments with the load balancer network function................. 47
Figure 6.3 Load balancer effectiveness 49

LIST OF TABLES

Table 4.1 Overview of the RTT experimental scenarios 29
Table 4.2 *Flow stats reply* TCP payload and Ethernet frame sizes 31

Table 5.1 Failover process results 43

Table 6.1 Flow ranges and installed rules 48
Table 6.2 Effect of control latency on data throughput 50

CONTENTS

1 INTRODUCTION **11**
2 LITERATURE REVIEW **13**
2.1 Wireless **13**
2.2 Control Plane Partition **14**
2.2.1 Physical distribution of the controller 15
2.2.2 Delegation of concerns to the forwarding plane 17
2.3 Discussion **18**
3 A PROPOSAL FOR THE SDN CONTROL PLANE WITH CELLULAR LINKS **19**
3.1 Additional Cellular Links **19**
3.2 Research Questions **21**
3.3 Testbed **22**
4 EVALUATING THE EFFECTS OF CELLULAR LINKS IN OPENFLOW **25**
4.1 Methodology **25**
4.1.1 Metrics 26
4.1.2 Message exchange scenarios 28
4.1.3 Input parameters 30
4.2 Experimental Results **32**
5 EVALUATING THE FAILOVER TO THE BACKUP CELLULAR LINK **36**
5.1 Methodology **36**
5.1.1 Experimental setup 37
5.1.2 Handover on Unmodified Open vSwitch 38
5.1.3 Handover on Open vSwitch with Lower Probe Interval 39
5.1.4 Handover with Multipath TCP 40
5.2 Experimental Results **43**
6 EVALUATING A NETWORK FUNCTION ON A CELLULAR CONTROL PLANE **44**
6.1 Methodology **44**
6.1.1 Network Function: Load Balancing 45
6.1.2 Experimental Setup 47
6.2 Experimental Results **49**
7 CONCLUSION **52**
REFERENCES **55**
APPENDIX A — RESUMO ESTENDIDO DA DISSERTAÇÃO **60**
A.1 Introdução **60**
A.2 Contexto e Motivação **60**
A.3 Arquitetura Proposta **61**
A.4 Avaliação **61**
A.5 Contribuições **63**
A.6 Trabalhos Futuros **64**
APPENDIX B — SOURCE CODE EXCERPTS **65**
B.1 Source Code of Changes to cbench **65**
B.2 Source Code of Changes to POX **67**
B.3 Source Code of Changes to Open vSwitch **68**
APPENDIX C — ACCEPTED PAPER AT IEEE ICC **69**

1 INTRODUCTION

Over the past few years, the Software Defined Networking (SDN) paradigm has gained unprecedented strength within the scientific community. The split of the control and forwarding (or data) planes is expected to increase development pace and to ease the job of network operation, through the separation of concerns and logically centralized control, respectively. As OpenFlow (MCKEOWN et al., 2008) consolidates as a *de facto* standard for the SDN southbound control plane protocol, the interoperability between compliant forwarding devices[1] and controller software allows for a vendor-neutral system. Considering these benefits, research has been devoted to applying SDN into other areas, including wireless networks.

As a consequence of the separation between planes, a dependency relation was created among entities which are now potentially physically separated – namely, the controller and switches. Therefore, partitions in the network can severely disrupt functionality (AKELLA; KRISHNAMURTHY, 2014). For instance, link failures can isolate devices from their corresponding control element, preventing them from being aware of policy updates, breaking the global visibility of the network and, in the absence of failover mechanisms, even render the device temporarily unusable.

A common strategy for preventing the control plane partition is to deploy the network controller as a physically distributed system. As control instances are spread over the topology, the isolation of forwarding devices from control on the event of network partitions becomes less likely. Nevertheless, additional challenges arise in the implementation of a logically centralized controller, as the system's distributed instances need to perform synchronization in a consistent and timely manner. An alternative strategy consists in delegating control plane tasks back to switches, attempting to attenuate the consequences of isolation rather than preventing it. However, this implies partially losing the advantage of the centralized network programmability brought by the SDN paradigm.

Motivated by these issues, this work explores an entirely different strategy, based on wireless links. More precisely, it evaluates the suitability of a control plane architecture featuring additional *cellular links* for improved resilience in face of network partitions in the control plane. The evaluation is focused on opportunities and limitations based on three aspects, namely: (a) the effects of cellular link characteristics on OpenFlow traffic, (b) the failover mechanism from the primary wired links to the backup wireless link, and (c) the operation of a network function that uses global network state awareness and can have a limited, configurable

[1]In the OpenFlow community, forwarding devices are often called simply *switches*, for short. The terms *forwarding device* and *switch* are used interchangeably throughout this document.

level of interaction between the forwarding device and the controller.

The main contribution of this dissertation is to provide a novel analysis of the operation of OpenFlow control plane traffic over cellular links. Experimental results indicate that, in spite of naturally providing lower bandwidth and higher latency than wired links, cellular links can still allow an OpenFlow-based network to maintain the operation of network functions under certain circumstances without requiring modifications to the underlying devices. Additionally, small software modifications on the forwarding devices and the use of recent network mechanisms such as Multipath TCP (PAASCH; BARRE et al., 2015) can greatly improve the handover to the backup control links. The suitability of this architecture to network functions is shown to depend on the level of reactivity required at the control plane. The most important benefit introduced by backup cellular links is to prevent control plane partition and, therefore, maintain a consistent, centralized, global view of the network, even if state awareness can be delayed by the additional latency of event notifications.

This dissertation is organized as follows: In Chapter 2 we present a literature review in SDN with wireless networks and control plane connectivity. In Chapter 3 we describe the proposed architecture. In the next three chapters we explore the opportunities and limitations of the proposed architecture by evaluating it from different perspectives. More precisely, Chapters 4, 5 and 6 are an attempt to answer important questions, respectively, *how is the behavior of control plane traffic affected by the characteristics of cellular links*, *how quickly is the control plane handed over to the backup link when a failure occurs* and *how well do network functions that rely on a snapshot of the network state behave on such an architecture*. Finally, Chapter 7 contains concluding remarks and perspectives for future work.

2 LITERATURE REVIEW

This dissertation features an architecture equipped with cellular network links for preventing the partition of the SDN control plane. In this chapter, related work is organized in two sections. In Section 2.1 we review work that concerns wireless networks in the context of SDN. In Section 2.2 we discuss proposals that share the same objective of the present work, which is to address the loss of connectivity of the control plane, by attempting either to prevent partition or to mitigate its impact. Finally, in Section 2.3 we compare the aforementioned works with the content of this dissertation.

2.1 Wireless

The promising benefits of Software Defined Networks – namely, separation of the network control logic from the forwarding devices, logical centralization of control, and enhancement of network programmability (KREUTZ et al., 2015) – have led to a great amount of research into applying this paradigm over various networking contexts. Wireless networks are no exception to this trend.

Yap *et al.* (YAP et al., 2010) state that wireless mobile network infrastructures are majorly proprietary and closed. The objective of their proposal is to decouple the infrastructure from the network services. The paper shows the potential benefits of using SDN and OpenFlow for centralized control of access points, simplifying tasks such as locating non-overlapping channels, reducing interference through power level adjustment, authenticating users and handling device mobility. The deployment of a prototype of the proposal is part of the OpenRoads project, later renamed to OpenFlow Wireless (OPEN NETWORKING FOUNDATION, 2011), from the Stanford OpenFlow Team. The evaluation of the OpenRoads prototype included measuring side effects of handover between infrastructure providers.

The criticism over the closed nature of wireless network infrastructure is echoed by Li *et al.* (LI; MAO; REXFORD, 2012). Although stressing the possibility of simplifying the design and management of cellular networks through SDN, the authors discuss scalability challenges that the new centralized control architectures should address, such as fine-grained measurements, frequent mobility and the growing number of subscribers.

Although it is not particularly related to SDN, the proposal of Zhou *et al.* (ZHOU et al., 2012) precedes the work of this dissertation in the idea of leveraging wireless links for extended connectivity. The additional links combined with reflective surfaces on the facility roof increase the trajectory space and allow for a greater number of simultaneous connections between pairs of nodes in a data center environment. In this scenario, the main concern for extending the connectivity is the additional bandwidth, not the prevention of partitions.

The work of Guimarães *et al.* (GUIMARAES; CORUJO; AGUIAR, 2014) features an extension for adding Media Independent Management into the OpenFlow protocol, so that link condition information can be used by the controlling mechanisms. For example, the controller could query for signal strength status of a given link, or request its power saving mode to be switched on/off. One of the objectives stated in the proposal is being oblivious to the link network technology. For the sake of making experiments easier, the setup featured wireless links solely in the forwarding plane. The control plane channels were established over Ethernet wires.

Aeroflux is a hierarchical network design for a software-defined WiFi network proposed by Schulz-Zander *et al.* (SCHULZ-ZANDER; SARRAR; SCHMID, 2014). The design features a global controller and subordinate near-sighted controllers, and the evaluation of an early prototype indicates reduction in the control traffic.

The idea of taking advantage of the locality of control operations is also explored outside the scope of wireless networks. Examples of papers featuring this approach are among the literature on control partition tolerance, presented in the following section.

2.2 Control Plane Partition

When partitions occur, network architectures with separate control and forwarding planes can suffer more severe compromise of functionality than traditional architectures (i.e. with integrated planes in a black box). The consequences of the loss of communication between a forwarding device and its controller include becoming unable to receive instructions for newly arriving flows and being unaware of updates in the routing rules – for instance, due to changes in access control policies.

Since version 1.1, the OpenFlow protocol (OPEN NETWORKING FOUNDATION, 2013) specifies two choices of behavior for conforming switches to perform while there is no communication with the controller: *fail-secure*, where messages towards the controller are dropped and the existing rules are applied until their own expiration times arrive, and *fail-standalone*, which means that the device operates with similar autonomy to an Ethernet switch.

The problem of control plane partition can be tackled with preventive measures. In particular, physically distributing the controller reduces the likelihood that a network partition can cause a forwarding device to be isolated from the control plane. The literature on this architectural strategy is discussed in Subsection 2.2.1. In contrast, there are proposals that do not prevent partitions, but instead attempt to limit the consequences of isolation. They consist of delegating control plane concerns to the forwarding devices, and are presented later, in Subsection 2.2.2.

2.2.1 Physical distribution of the controller

In the literature, the term *distributed controller* often refers to architectures designed for server clusters. In these proposals, control instances depend on some underlying mechanism for sharing state, such as key-value stores or shared memory. Examples include open-source controllers such as Opendaylight (OPENDAYLIGHT, 2015) and ONOS (BERDE et al., 2014), proprietary software – e.g. HP VAN SDN (HEWLETT-PACKARD DEVELOPMENT GROUP, 2013) – and also prototypes of architectural designs such as PANE (FERGUSON et al., 2013) and ElastiCon (DIXIT et al., 2014). The main goal of employing distribution in these proposals is to leverage parallelism. This type of architecture also improves fault tolerance, in particular, when controller instances in the cluster fail. As instances are often strongly connected to each other, partitions within the cluster are unlikely. However, an edge link failure could isolate a forwarding device outside the cluster form every single control instance.

Greater attention will be directed towards architectures that consider control instances spread through the topology, as these help avoiding the isolation between forwarding devices and controllers. The term *distributed controller* will be used throughout this document in reference to this type of proposal. In fact, positioning of distributed instances is a topic of research on its own (ZHANG; BEHESHTI; TATIPAMULA, 2011; BEHESHTI; ZHANG, 2012; HELLER; SHERWOOD; MCKEOWN, 2012; BARI et al., 2013; HU et al., 2013; MULLER et al., 2014; SALLAHI; ST-HILAIRE, 2015).

One of the first proposals implementing a SDN platform control as a distributed system was Onix (KOPONEN et al., 2010). With reliability and scalability in mind, the Onix architecture features one or more instances running on a cluster of physical servers. However, the challenge of keeping consistency increases as both the number of distributed instances and update rate grow. A trade-off between performance and scalability can be configured by choosing one of two distribution mechanisms for a given network state: a memory-based one-hop DHT or a replicated transactional database. The implementation was described at the time as a production beta trial for commercial deployment, and later on it was the base for the controller used by Google in B4 WAN (JAIN et al., 2013). The paper has an early glimpse on exploring node locality, but only as an example of use of its instance aggregation mechanism.

In contrast, the authors of HyperFlow (TOOTOONCHIAN; GANJALI, 2010) propose localizing decisions to individual controller instances, both for partition tolerance and for smaller control plane latency. State synchronization is obtained through a publish/subscribe mechanism atop a distributed file system. This choice of design is expected to maintain a network-wide view through selective publishing of events, based on some observations over the characteristics of network state. HyperFlow was implemented as an application to be executed over NOX (GUDE et al., 2008) instances. It requires the concurrent applications to be modified so as to tag events that affect their own state – that is, the channels that they must subscribe to. Moreover, applications cannot rely on temporal ordering of events except those targeting the same network entity (switch, link or host). Further still, the number of effective events must be bounded by the number of entities, and not by the number of flows. Otherwise, the overhead would be prohibitive. Other modifications to the applications are recommended so as to take into account the number of controllers. For instance, measurement applications should be aware of the local controller assignments and have their jobs partitioned, in order to avoid the linear growth of queries towards the forwarding devices.

Kandoo (HASSAS YEGANEH; GANJALI, 2012) takes an alternative route by refraining from keeping global-view consistency between instances. Instead, it allows the applications to leverage the locality of some control plane operations by offering a two-level controller hierarchy tree. Applications that depend on awareness of network-wide state must run on the root instance, while those that can operate locally (e.g. enforcement of local policies, LLDP) should be deployed on control instances that are positioned closer to the forwarding devices. No implicit synchronization between the instances is performed and, therefore, root applications can only communicate with the ones at the lower level of the control tree through explicit event subscription.

In the follow-up paper Beehive (HASSAS YEGANEH; GANJALI, 2014), the authors define a programming abstraction for the control application, aimed at taking advantage of the locality of distributed instances. Each event handler in the application should declare the entries of the state dictionary that it depends on. Based on this declaration, the Beehive framework dynamically assigns functions to specific controller instances, trying to deploy them closer to the switches that originated these events. If the state dependencies allow, then a single function handler can be split among the controllers, each of which running independently. The objective is to take advantage of distributed instances while trying to limit as much as possible the communication for state synchronization. However, functions that depend on network-wide view are still deployed in a single instance and depend on event propagation.

2.2.2 Delegation of concerns to the forwarding plane

The extreme case of leveraging locality is to delegate a function to the forwarding plane altogether, so as to maintain its functionality even under control plane partition. As discussed in the beginning of Section 2.2, OpenFlow specifies an optional fallback mechanism for a forwarding device that lost its connection with the controller. It consists in reverting to usual Ethernet switch behavior, the so-called *fail-standalone* mode on a *hybrid switch*. For instance, the devices built for Google's B4 WAN (JAIN et al., 2013) are capable of running routing protocols directly, but they also have a software layer for receiving OpenFlow commands from a remote controller and translating them into the forwarding hardware.

The authors of DIFANE (YU et al., 2010) propose the modification of some devices into *authority switches*. The controller proactively partitions the set of specified policies and installs lower-level routing rules in those devices, using wildcard matching when applicable. Unmodified ingress devices handle their table misses by redirecting the packet from the offending flow to an authority switch – the one assigned by the policy partition. Additionally, the authority switch installs a rule matching this flow in the incoming device, keeping the process within the forwarding plane. The consequences of failures in the control plane are diminished as the authority switches can handle a limited set of control operations.

In the work of Liu *et al.* (LIU et al., 2013), routing connectivity can be ensured at the forwarding plane by using small and fast operations that store state in the packet headers. This mechanism is based on the properties of Directed Acyclic Graphs, by which a finite set of *all-edges-outward* operations over the nodes guarantees a directed path to the destination node. The link inversion operations are informed to the adjacent nodes through the packet headers. Assuming that there is connectivity towards the packet destination in the forwarding plane, packet delivery is maintained even without control plane communication. This means that forwarding is resilient to any combination of data link failures that does not isolate the source and destination nodes – in contrast, for example, to pre-computed backup paths, where the number of required backup rules is usually proportional to the set of concurrent failures that can be tolerated. However, updates in policies still depend on control plane connectivity.

2.3 Discussion

The proposals described in Section 2.1 have the merit of employing SDN as a framework to control the infrastructure of wireless and mobile networks or as a tool to improve provision of services. However, those papers do not address using wireless networks for tolerance or prevention of control plane partition.

Work discussed in Section 2.2 presents two groups of approaches regarding this issue, in Subsections 2.2.1 and 2.2.2. The proposals in Subsection 2.2.1 address control plane partition by deploying a logically centralized controller as a physically distributed system, eliminating the single point of failure. However, the trade-offs of the CAP theorem (consistency, availability and partition tolerance) apply to SDN as well (PANDA et al., 2013). This physical distribution is characterized by a compromise between the complexity of keeping state synchronized among the controller instances and the loss of global view of the network.

The proposals favoring state consistency between control instances resort to a variety of sharing mechanisms, usually depending on an underlying external system. However, the use of such a system raises its own issues (HASSAS YEGANEH; GANJALI, 2014). First, the control application has no influence over the physical placement of state shares in the distributed store. Secondly, interfacing between the controller and the external storage system incurs overheads on the control plane. Finally, the storage system itself must be managed as well.

A common way to diminish the impact of losing global state awareness is to exploit locality of control plane operations. Research towards identifying particular state distribution characteristics of a network control plane is underway (SCHMID; SUOMELA, 2013; AKELLA; KRISHNAMURTHY, 2014; CANINI et al., 2015) and it could enable partition tolerance for a growing number of applications. However, as long as there are network functions that are intrinsically dependent on a global and consistent network view, a logically centralized controller is required. If such a controller is physically distributed, this means that state must be consistent among instances.

When the loss of connectivity between forwarding devices and their control units cannot be avoided, some of the consequences can be mitigated by mechanisms that delegate concerns to the forwarding plane, as discussed in Subsection 2.2.2. Yet, these techniques are a step back in the trend of keeping the forwarding devices simple and centralizing network programmability.

In contrast to related work, the proposal described in this dissertation requires no modifications to the forwarding devices, to the controller or to the OpenFlow protocol for keeping control plane connectivity in the event of link failures.

3 A PROPOSAL FOR THE SDN CONTROL PLANE WITH CELLULAR LINKS

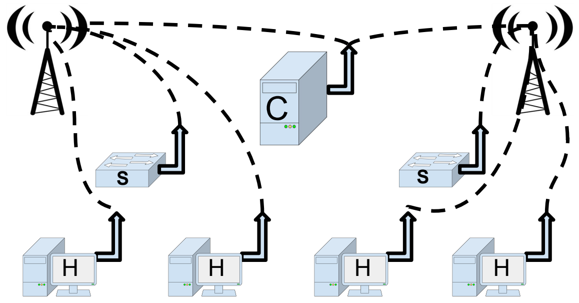
Taking into consideration the importance of maintaining connectivity between forwarding devices and controller, we build on our previous work (PETRY; SILVA; BARCELLOS, 2015) in describing and evaluating a novel SDN control plane architecture where additional cellular links (e.g. 4G, LTE) are employed as a backup resource for the control plane.

This architecture can provide partial functionality and network management activities under failures in the primary wired control links, transparently to the network elements and without incurring in high costs on hardware. In Section 3.1 we present three possible scenarios for adding cellular links in SDN, the latter of which is the focus of this work. In Section 3.2 we briefly describe the three research questions that guide the evaluation of the chosen setting. A common testbed and a set of assumptions that apply to all of the questions is presented in Section 3.3.

3.1 Additional Cellular Links

There are three main settings in which additional cellular links could be used to extend the network: a) for shared forwarding plane traffic and physically in-band control plane traffic, b) for east/westbound control plane traffic among controller instances, and c) for southbound control plane traffic between forwarding device and controller.

Figure 3.1: Cellular links in Software Defined Networks for shared forwarding plane traffic and physically in-band control plane traffic. *C* stands for controller, *S* for switch and *H* for host.

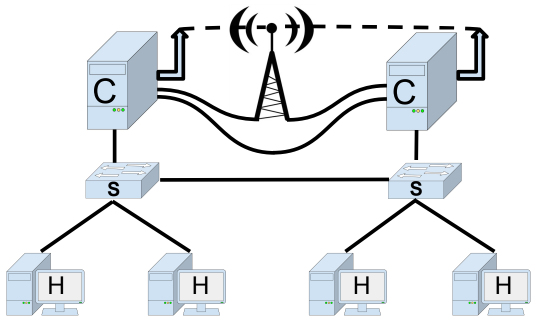


Source: by author (2015).

The first alternative, shown in Figure 3.1, has already been suggested in some of the works discussed in Section 2.1 (YAP et al., 2010; LI; MAO; REXFORD, 2012; GUIMARAES;

CORUJO; AGUIAR, 2014). However, as link failures are concerned, this design features shared fate between control and forwarding planes, and it is not in the scope of this work.

Figure 3.2: Cellular links in Software Defined Networks as backup for east/westbound control plane traffic among controller instances. *C*: controller, *S*: switch and *H*: host.



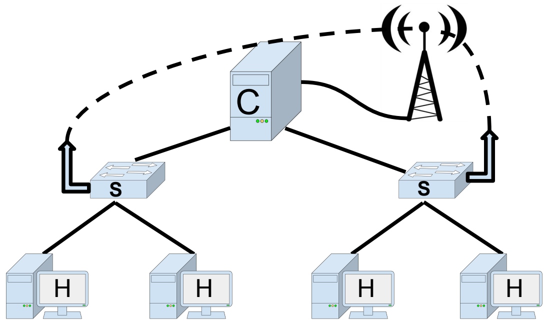
Source: by author (2015).

Figure 3.2 illustrates the second alternative. Albeit the architecture is promising for preventing partition of control instances, it still depends on the convergence of an east/westbound standard, as well as on the evolution of open distributed controllers. As discussed in Section 2.2.1, previous work on controllers with distributed instances either consists of closed-source solutions or relies on non-standard synchronization frameworks such as distributed file systems or key-value pair stores. Certain restrictions in control application behavior and policy decisions towards state synchronization have already been noted (TOOTOONCHIAN; GANJALI, 2010), notably for scalability reasons related to event handling and device probing. Although this possibility is not explored here, the use of backup cellular links for keeping state synchronized among separate controller instances has good potential inasmuch as research on distributed controller state matures.

The third alternative, shown in Figure 3.3, is to extend the control plane connectivity through additional wireless links between the forwarding devices and the controller. That is, the forwarding element has two paths towards the controller: a primary wired link, in which control plane traffic is carried under normal circumstances, and a backup 4G link from a different network, serving as a contingency resource should a failure occur in the primary channel. This approach is the focus of this dissertation, which is structured according to a set of research questions to be answered.

Each question is presented in a separate chapter, with methodological aspects specific to the experiment designed to answer it. The following section describes the research outline by briefly introducing each question and showing a common ground on the methodology and the testbed for all of them.

Figure 3.3: Cellular links in Software Defined Networks as backup for southbound control plane traffic between forwarding device and controller.



Source: by author (2015).

3.2 Research Questions

The aim of this research is to answer the questions: **Q1.** how the behavior of the control plane is affected by characteristics of cellular links; **Q2.** how quickly the control plane is handed over to a backup link when a failure occurs; and **Q3.** how well do network functions that rely on a snapshot of the network state behave on such an architecture.

The first question is treated with a set of experiments measuring the mean response time of interactions between a forwarding device and a controller. The objective is to determine the effects of the link characteristics on OpenFlow traffic for different packet sizes and a range of simultaneous message exchanges. In short, the response time of a message exchange is the elapsed time between sending an OpenFlow message from one end of the connection to the other and receiving its corresponding reply, completing a round trip. The mean RTT (Round-Trip Time) of message exchanges started at both ends of the control plane channel is used for comparison between wired and wireless links. The metric is based on the output of *cbench* (TOOTOONCHIAN et al., 2012), a tool developed for SDN controller benchmarking. Taking the network control out of forwarding devices and moving it into a logically centralized entity has performance implications that are recognized by the SDN community, particularly concerning the controller resource consumption (YU et al., 2010; CURTIS et al., 2011) and control channel overheads (YU et al., 2013; CHOWDHURY et al., 2014; ISOLANI et al., 2015). In the proposed architecture, it can be assumed that the cellular link represents a stricter performance bottleneck on control plane traffic than the controller hardware itself. Therefore, the same metrics of controller benchmarking are used for evaluating the cellular link features – such as higher expected latency and lower available bandwidth – over OpenFlow. A detailed description of the experiment and the results are presented in Chapter 4.

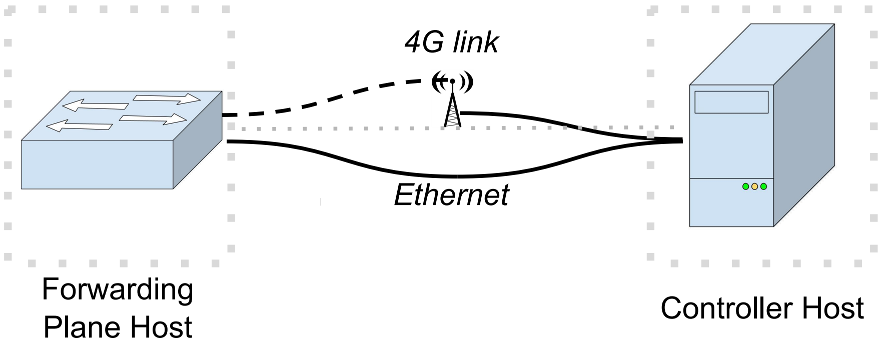
The second question concerns the handover procedure for the control plane channel. The forwarding element is multihomed, thus being able to reach the controller through two disjoint paths: a primary wired link and a backup 4G link. A failure is injected in the primary link, and three handover procedure alternatives are evaluated. First, with an unmodified Open vSwitch (OPEN VSWITCH, 2015). Second, with a slightly modified version of the software switch for a smaller inactivity probe interval. Finally, with the original switch and a Multipath TCP connection. The failover metric is defined as the elapsed time between the last control message received at the controller through the defunct primary link and the first message arrived through the backup link. Chapter 5 discusses the benefits, drawbacks and requirements for each of the three alternatives.

The third question is answered by experimenting an application that operates over a snapshot of the network state on the proposed architecture. A very popular choice for this kind of application in the literature is load balancing – examples are shown in Chapter 6. Therefore, this function is implemented with a strategy that mixes reactive installation of rules and proactive routing of flows, based on a scheme proposed in the work of Wang *et al.* (WANG; BUTNARIU; REXFORD, 2011). The behavior of the load balancer application is observed using different statistics collection rates and flow rule granularities. The efficiency of load balancing is evaluated by measuring the server port usage, while the timeliness of packet delivery is obtained by observing the forwarding plane throughput.

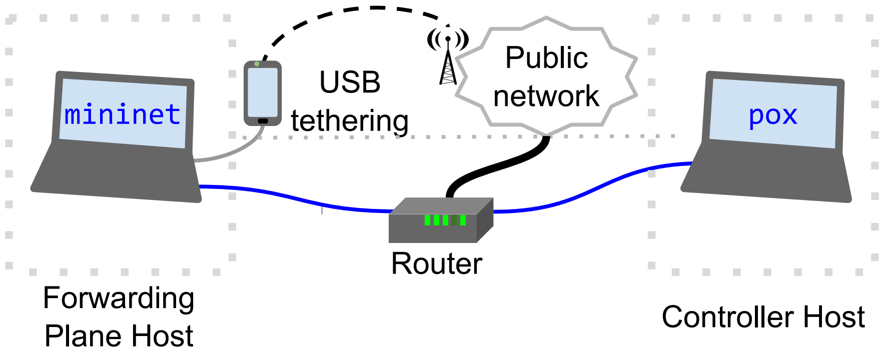
3.3 Testbed

The experiments performed to answer the three research questions share a common testbed, illustrated in Figure 3.4, and a set of assumptions. In each of the experimental settings described, the *Forwarding Plane Host* is a 2 GHz Core 2 Duo machine with 4GB RAM running VMware Player 6 on Windows 7 32 bits. The virtual machine image receives a single processor core and has 1GB RAM available. The Mininet Virtual Machine (LANTZ; HELLER; MCKEOWN, 2010) image version 2.1.0 was used, with either the bundled Open vSwitch 1.9.0 or its recompiled version with altered lower bound for the inactivity probe. Where applicable, *cbench* is executed on this same virtual machine image.

Figure 3.4: Basic experimental environment for this dissertation. The illustration in 3.4a semantically represents our proposed architecture with the OpenFlow switch, controller and the partially disjoint paths via 4G LTE and Ethernet. The implemented testbed is shown in 3.4b.



(a) Abstract illustration



(b) Implemented testbed

Source: by author (2015).

The *Controller Host* runs POX (POX, 2015) 0.2.0, *carp* build, on a 1.8 GHz (x 4) Core i3 with 4GB RAM, on Ubuntu 14.04 64 bits. The machine is connected to a router through an Ethernet cable. The router has a public fixed IP address, and it is configured to forward incoming traffic with destination port 6633 (standard for an OpenFlow controller) towards the same port number on the *Controller Host*.

The *Forwarding Plane Host* can reach the *Controller Host* either by a 4G LTE cellular network accessed via USB tethering through the Internet or by a local wired Ethernet connection to the same router. The virtual machine has bridged access to both physical network adapters. The control plane is deployed *out-of-band*, which is a typical choice in SDN deployments (PANDA et al., 2013; JAIN et al., 2013). This means that control traffic is routed and forwarded through a legacy network and is not itself managed by OpenFlow.

The experiments were conducted in the central area of Porto Alegre, Brazil, starting around 11 PM local time, due to the good 4G LTE signal quality found during this period of the day. The cellular service was obtained by subscription to one of the top four mobile providers operating in Brazil. The cellular reception signals of the carrier ranged around -75dBm and -85dBm. Since the 4G service is obtained via the mobile provider base station, the experiments performed over the cellular link are subject to a small additional latency as traffic reaches the home router through the public Internet. Meanwhile, traffic over the wired channel can be delivered directly through the home router.

The deployment of Multipath TCP required support in the kernel at both ends of the control connection. This was achieved by installing the *linux-mptcp* module both on the Mininet image and on the *Controller Host* operating system, provided by (PAASCH; BARRE et al., 2015), with *linux-image-3.14.0-89-mptcp* kernel.

The following two assumptions are considered in the evaluation: (i) there is sufficient signal availability on the site of the hosts (i.e. forwarding devices) connected to 4G interfaces, and (ii) a host can reach the controller listening interface through the 4G links.

The first assumption is reasonable, given that signal coverage on metropolitan areas is widely available. This work is focused in the three questions presented in Section 3.2, addressing particular aspects of an architecture with backup cellular links for the control plane, assuming that the signal quality is good. As such, this work is a novel stepping stone for evaluating the suitability of this architecture. Assessing the characteristics of signal quality throughout hours and days and comparing the quality of service offered by different providers are very important measures for production deployments. However, both aspects are specific to the particular region of deployment, and thus, are out of the scope of this dissertation.

The second assumption could raise concerns about using public networks for control traffic. However, it is feasible to apply the security measure recommended by the OpenFlow protocol specification (OPEN NETWORKING FOUNDATION, 2013), which consists in securing the control channel with SSL.

The testbed we have available and that we used throughout this evaluation has a very limited scale if compared to the infrastructure of some production networks. However, we believe that it still allows us to obtain valuable insights towards answering our research questions, which can be extended into conclusions that also apply to larger scale environments.

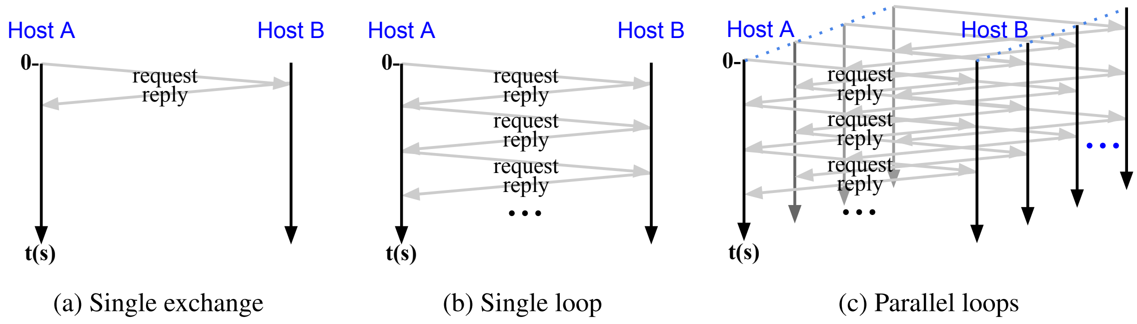
4 EVALUATING THE EFFECTS OF CELLULAR LINKS IN OPENFLOW

Some features of cellular links – namely, latency and bandwidth – are expected to be limiting factors to be dealt with in actual control plane network deployments. The following set of experiments stresses the communication between controller and forwarding devices, in an attempt to measure how strong this limitation is in comparison to wired connections and, therefore, walk towards answering the first question of this research: **Q1.** *How is the behavior of control plane traffic affected by the characteristics of cellular links?*

4.1 Methodology

With OpenFlow, the control plane communication is achieved through a TCP connection between a forwarding device and a controller. To understand the effects of the underlying media on control traffic, response times are measured for message exchanges between both ends.

Figure 4.1: Terminology for message exchanges in the RTT experiment. A *single exchange* (a) refers to a pair of messages exchanged between two hosts. A *single loop* (b) is a sequence of consecutive *single exchanges*. A number of *parallel loops* (c) can take place simultaneously between the hosts.



Source: by author (2015).

The message exchange loop used in this experiment is the same performed by *cbench*, a tool for controller benchmarking (TOOTOONCHIAN et al., 2012). Three terms that help describing the methodology are now defined, as they will be used throughout the text. Figure 4.1 illustrates each of them. The term *single exchange* (4.1a) is used to describe one round trip, where *host A* sends a message to *host B*, which in turn replies with a second message. On a *single loop* (4.1b), *single exchanges* are performed in series. Each time that *host A* receives a reply from *host B*, it starts another exchange by sending a new message. Additionally, many *single loops* can be performed asynchronously. They will be referred to as *parallel loops* (4.1c).

The methodology of this experiment is described according to the following structure: in Subsection 4.1.1, the metrics for the RTT measurement are presented in detail. Then, three scenarios are described in Subsection 4.1.2, featuring the differences in the message pair of single exchanges and the tools employed to perform the experiment. The input parameters are discussed in Subsection 4.1.3.

4.1.1 Metrics

The diagrams in Figure 4.2 illustrate the parallel loops for each of the three experimental scenarios. Messages with varying payload are written in capitals and are represented by thicker arrows. A total of n parallel loops occur simultaneously over 30 seconds, split into slots of one second. The single exchanges are not bounded within a given time slot. For instance, the starting host can send a message near the end of time slot t_i and receive the reply already during time slot t_{i+1}. In any case, an exchange is considered to belong to the time slot in which the response is received back at the initiating host.

Let $r_t[e]$ be the number of replies received by the starting host within single loop e during time slot t. Let $\bar{r}_t$ be the mean number of replies received within the n parallel loops during time slot t, as in Equation (4.1).

$$\bar{r}_t = \frac{1}{n}\sum_{e=1}^{n} r_t[e] \tag{4.1}$$

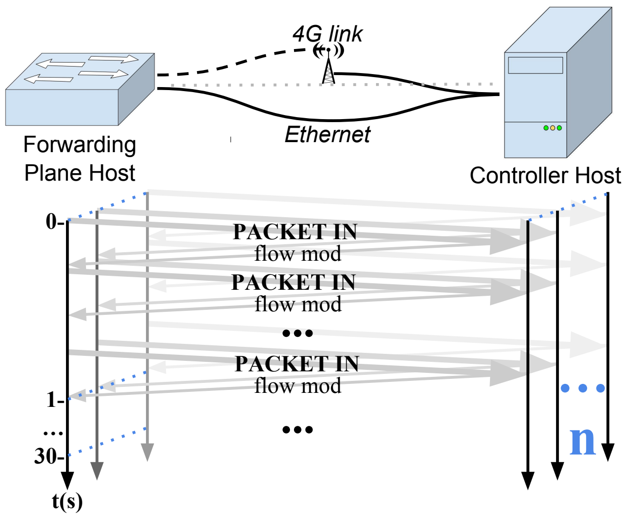
The mean round-trip time during a single slot is given by the inverse of the mean number of replies received during that slot (4.2).

$$RTT_t = \frac{1}{\bar{r}_t} \tag{4.2}$$

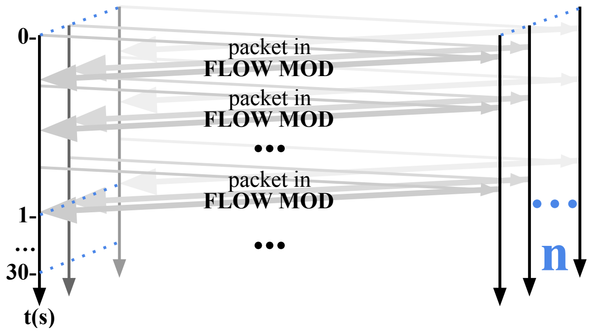
The average exchange times featured in the plots (4.3) are measured over 30 consecutive time slots of one second each, based on the number of replies received within each slot per loop. For the error bars, the chosen confidence interval is 99%, using Student's *t* distribution with 29 degrees of freedom ($T = 2.756$).

$$\overline{RTT} = \frac{1}{30}\sum_{t=1}^{30} RTT_t \tag{4.3}$$

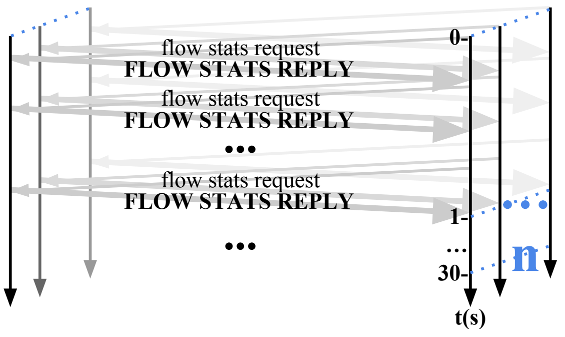
Figure 4.2: The *Forwarding Plane Host* reaches the *Controller Host* either by a wired connection or by a 4G link. In each single exchange of conversations initiated at the *Forwarding Plane Host* (4.2a, 4.2b), *packet in* messages are sent with *cbench*, and the POX controller at the opposite host responds with *flow mods*. In the single exchanges of the opposite sequence (4.2c), the control application at the *Controller Host* sends *flow stats requests*, and a single switch on *Mininet* reacts with *flow stats replies*. Each of the three timelines illustrate n parallel loops, over 30 time slots of one second for the experiments a, b and c. Messages with varying payload are written in capitals and are represented by thicker arrows.



(a)



(b)



(c)

Source: by author (2015).

This metric is aimed at assessing the average round trip time of control plane traffic with each underlying link media, featuring some variations in the exchange pattern through the choice of input parameters. As parallel loops take place asynchronously, it is often the case that some of the single loops are able to perform more exchanges than others in a given time slot. Occasionally, a single loop may even remain in starvation for a while. The interleaving among parallel loops is not in the scope of this evaluation. Therefore, the average elapsed time for an exchange to complete from all parallel loops is taken as an indicator of the performance of control traffic over a given channel.

4.1.2 Message exchange scenarios

The performance of the control plane message exchange on a cellular link is influenced particularly by the higher latency and the lower bandwidth obtained with this media. Therefore, the experiments consist in filling the channel either with more parallel loops or with increasingly larger packets. Additionally, the characteristics of the evaluated architecture and the testbed require considering differences between the upstream and downstream data rates, which are usually asymmetric in LTE service (LTE, 2015). Most clients served by retail mobile providers are also behind NATs, making it difficult for an external server to directly initiate communication, as unidentified packets tend to be filtered out (CHEN et al., 2013). As the OpenFlow connection is established by the switch to a fixed IP address assigned to the controller (OPEN NETWORKING FOUNDATION, 2013), this means that the cellular interface must be attached to the *Forwarding Plane Host*. Consequently, messages sent by the *Controller Host* are always on the downstream direction, which has a higher data rate, and messages sent by the *Forwarding Plane Host* have a more restrict data rate in the upstream direction.

In order to assess the impact of these features, three evaluation scenarios are proposed, based on the message exchange loop described earlier. As *cbench* is intended as a controller benchmark tool, it poses itself as a switch from the controller's point of view, sending *packet in* messages and measuring the mean response time. Two scenarios follow this same scheme, featuring message exchanges initiated at the *Forwarding Plane Host*. In the first one, the channel is filled with increasingly larger messages upstream in each execution, while in the second one the message length variation occurs downstream. The third scenario is aimed at evaluating message exchanges started at the opposite end of the connection. Since the cellular interface is fixed at the *Forwarding Plane Host* and the *packet in - flow mod* pair cannot be flipped, the statistics collection message exchange is used instead. Table 4.1 summarizes the three scenarios.

Table 4.1: Overview of the RTT experimental scenarios

Exchange diagram	Figure 4.2a	Figure 4.2b	Figure 4.2c
RTT plot	Figure 4.3a	Figure 4.3b	Figure 4.3c
Generating host	*Forwarding Plane Host*	*Forwarding Plane Host*	*Controller Host*
Message pair	*packet in* *flow mod*	*packet in* *flow mod*	*flow stats request* *flow stats reply*
Message with varying size	*packet in*	*flow mod*	*flow stats reply*

Source: by author (2015).

***Packet in* and *flow mod*.** This message pair is one of the fundamental OpenFlow processes. It consists in the reactive rule installation procedure, triggered by the arrival of a flow that is not currently matched by any rule at the forwarding device table. After processing the message according to the policies programmed into the control applications, the controller replies with a message instructing the device what to do with the flow or with that individual packet. This message could be a *flow mod*, which tells the device to install a rule in its table.

In order to change the size of the *packet in* messages sent upstream by the *Forwarding Plane Host* for the first scenario, this work includes an extension to *cbench* that allows configuration of the payload length of generated forwarding plane packets embedded into the *packet in*. The source code differences are displayed in Appendix B.1. The controller application reacts to each received *packet in* by sending an actionless *flow mod*.

In the second scenario, the size of the messages sent by the *Controller Host* is adjusted by attaching an action set to the *flow mod*. This set is comprised of a number of 8-byte *output* actions and a single 4-byte *strip VLAN* instruction, obtained with a small modification in POX for removal of four additional padding zeroes (as shown in Appendix B.2). This is to assure that the payload size is consistent among every RTT experiment. In these two scenarios, control traffic is generated by *cbench* with *N* parallel loops, which corresponds to one of the parameters of the tool that defines the number of emulated switches.

***Flow stats request* and *reply*.** The message exchange for the third scenario is started at the controller. It is part of the statistics collection mechanism. The controller queries the statistics of each flow that has a rule installed at the forwarding device flow table with a *flow stats request* message. The device answers with a *flow stats reply*, including statistics about each individual flow, such as the rule duration and the number of transferred packets and bytes.

The control application manages the size of the device response message. Statistics for each installed rule have a fixed length. Thus, the size of the *flow stats reply* is a function of the number of rules installed by the controller into the flow table of the device. The *Forwarding Plane Host* runs an emulated network environment within Mininet. As the focus of the experiment is the control plane traffic, the forwarding plane topology is kept as simple as possible, in order to limit the processing overhead. Therefore, a single switch is deployed. Before starting measurements, the control application proactively installs a set of 1 to 10 rules, each with a different match. In this experiment, the choice is to use distinct matching values for the data link layer protocol header. The size of this rule set is defined to achieve the desired response packet length. After the installation of the rules, the control application sends *N* concurrent *flow stats requests* and accounts for the replies sent by the emulated switch.

It is worthy of note that the change of the initiating host and message pair required for the third scenario implies in limitations in the experiment, as additional processing is performed both at the *Forwarding Plane Host* and *Controller Host*. In the first two scenarios the *Forwarding Plane Host* executes *cbench*, which is specialized for generating *packet in* messages and accounting for the replies. Additionally, it can perform basic OpenFlow interaction for establishing a number of concurrent TCP control connections and answering to eventual controller queries with dummy answers. In contrast, the third scenario features a full-featured software switch emulated in Mininet. As for the *Controller Host*, the control application running in POX for the first two scenarios simply replies to a *packet in* from any control plane connection with an empty *flow mod*. In the third scenario, the application manages the *N* parallel loops and the measurement of replies, however maintaining a single control connection with the emulated switch. While these limitations cause a lower performance on the hosts, the results show that the control channel media is the stricter bottleneck, particularly as the values of the input parameters increase.

4.1.3 Input parameters

The number of parallel loops is the first input parameter. In the result plots, each choice of value is represented on a separate series. The second variable is the size of the payload of the messages, corresponding to the horizontal axis of the plots.

Increasing the number of parallel loops or the payload of the messages going in one direction result in filling the control channel. The difference lies in the number of messages traveling the other way around. For instance, changing the number of loops from *N* to *2N* or doubling the payload of messages upstream would seem to equally fill the upstream bandwidth. However, each single loop has synchronous exchanges. The choice of doubling the number of parallel loops incurs in doubling the number of downstream messages, while doubling the payload does not. In summary, an experiment with double the number of loops is more sensitive to link latency than an experiment with double the payload.

In each of the three scenarios, the number *N* of parallel loops ranges from 32 to 160. The generating host initiates the *N* loops. For each completed single exchange inside a loop, a new one is started. Within a warm up period of one second, the number of parallel loops stabilizes as *N*.

Table 4.2: TCP payload and Ethernet frame size of *flow stats reply* as a function of the number of flow rules installed. In the other two RTT experiments, *packet in* or *flow mod* messages are set to the same sizes.

Flow rules	r	1	2	3	4	5	6	7	8	9	10
TCP payload	$p(r)$ (4.4)	108	204	300	396	492	588	684	780	876	972
Ethernet frame	$f(r)$ (4.5)	178	274	370	466	562	658	754	850	946	1042

Source: by author (2015).

The range of packet sizes for each RTT experiment is based on the payload of *flow stats replies* covering from 1 upto 10 flow rules, as shown in Table 4.2. The OpenFlow header of this message has 12 bytes, and statistics for each rule have 96 bytes. The TCP payload p as a function of the number of rules r is, therefore:

$$p(r) = 96r + 12 \tag{4.4}$$

The bottom row of Table 4.2 shows the layer 2 Ethernet frame length, including additional 70 bytes: 14-byte Ethernet, 20-byte IP and 32-byte TCP headers and a 4-byte Ethernet Frame Check Sequence. Summing up the OpenFlow header, the frame size f as a function of the number of rules r is:

$$f(r) = 96r + \underbrace{\overbrace{14}^{\text{Ethernet}} + \overbrace{20}^{\text{IP}} + \overbrace{32}^{\text{TCP}} + \overbrace{12}^{\text{OF}}}_{\text{Headers}} + \underbrace{4}_{\text{FCS}} = 96r + 82 \tag{4.5}$$

As described earlier, the payload is controlled (i) with the additional *cbench* parameter in the switch-initiated exchange with varying *packet in* length, (ii) with an attached set of actions by the control application when varying *flow mod* size and (iii) with preliminary installation of flow rules in the statistics exchange.

The effects of the underlying media on control plane traffic generated in these scenarios will be compared between the Ethernet and the 4G LTE link in the following section. While it is expected that the communication over the wireless link will have a higher latency, the intent of this experiment is to identify whether there is a stable, bounded time for these exchanges and what are the effects of having concurrent exchanges with varying message sizes. An example of this scenario is to have several forwarding devices simultaneously performing such interactions on a shared link. Furthermore, cellular links often have different upstream and downstream connection characteristics. Comparing a message exchange by varying the packet size in one direction at a time can help assessing the impact of this difference.

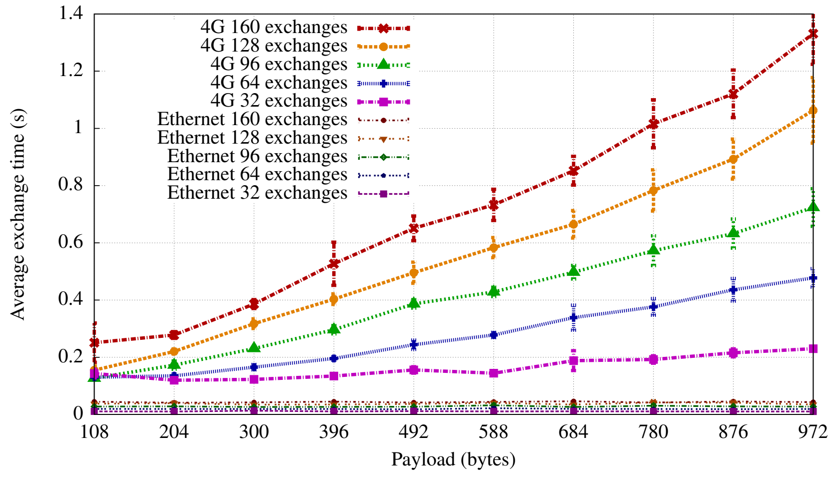
4.2 Experimental Results

As shown in the first two plots of *packet in* and *flow mod* exchange (Figures 4.3a and 4.3b) the cellular link in this experimental environment has a base RTT of around $120ms$. While lower latency values for operations such as *ping* and file download over 4G LTE have already been published (CHEN et al., 2013) – with mean RTTs below $100ms$ – this difference is in great part due to the characteristics of the carrier networks and the service offered in each region. This is supported by consistent results between OpenFlow message exchanges with small payloads and *ping* RTTs, both performed in the experimental testbed described here. In any case, this lower bound is significantly higher than the one achieved with a direct Ethernet wire.

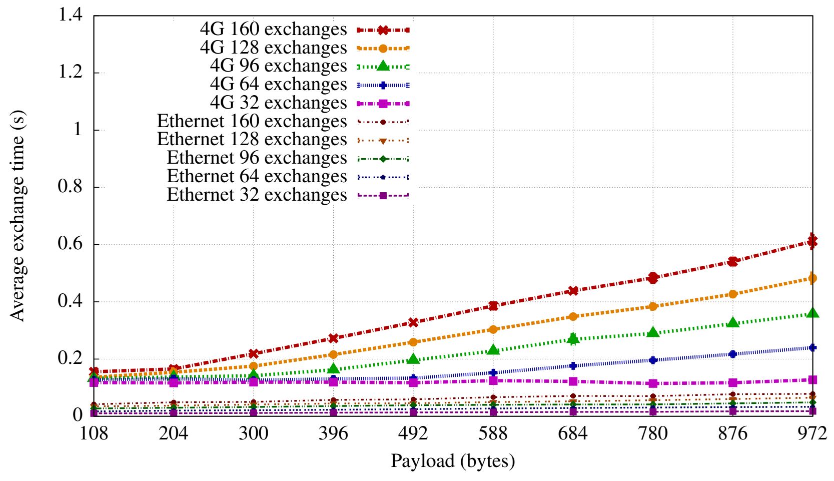
Additionally, the RTT increases as the channel bandwidth is filled either by increasing packet sizes or by adding parallel loops. It can be seen in the *4G 32 exchanges* line of Figure 4.3b that the mean latency is consistent among all payload sizes along the horizontal axis. Also, the same line in both Figures 4.3a and 4.3c shows but a slight slope upwards as the payload is increased. However, as the number of parallel loops is linearly increased, the RTT line slope gradually becomes steeper. The occupation of the channel bandwidth causes some messages to be buffered and delayed. In fact, when using smaller payloads – left-hand size of the horizontal axis in each plot – the mean RTT for any number of parallel loops over the same media is quite similar.

One important consequence of the impact of bandwidth occupation on the message RTT is that, when the volume of data is large enough, a same volume has different latency values upstream and downstream, proportionally to the bandwidth difference between them.

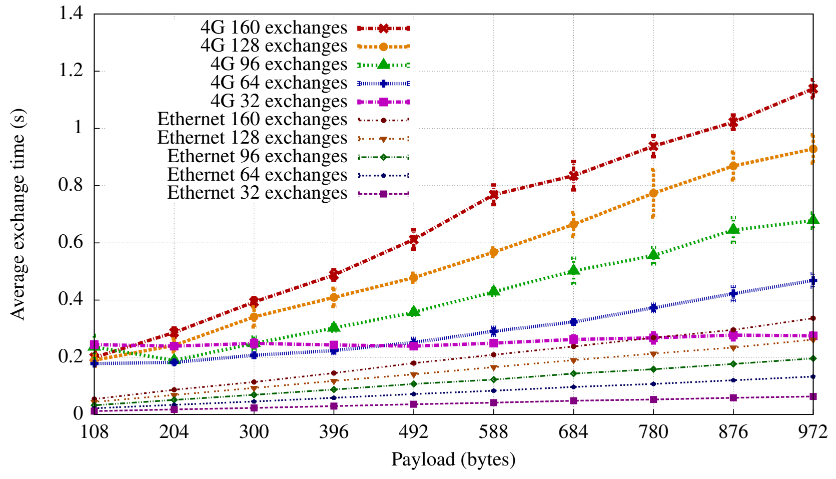
Figure 4.3: Average RTT for both media in each scenario.



(a) Varying *packet in* and fixed *flow mod*



(b) Fixed *packet in* and varying *flow mod*



(c) Fixed *flow stats request* and varying *reply*

Source: by author (2015).

This is supported by the comparison between Figures 4.3a and 4.3b. The only relevant difference between both scenarios is the direction in which the payload is changed – upstream with *packet in* in Figure 4.3a and downstream with *flow mod* in Figure 4.3b. All the 4G lines in both plots are lower bounded, as discussed eariler. However, as either the payload or the number of parallel loops increase, the lines gradually indicate the $1/2$ ratio between the upstream and downstream bandwidth reflected into the RTTs.

In Figure 4.3c, the RTT behavior of the *flow stats* exchange over the cellular link becomes similar to the first exchange with varying *packet in* size (Figure 4.3a) as either the number of parallel loops or the payload are increased. Both experiments fill up the channel with *Forwarding Plane Host* messages towards the *Controller Host* over the upstream connection. The lower bound, however, is quite higher in the *flow stats* exchange, with the RTT around 200ms.

The statistics exchange experiment has overheads such as having an emulated switch on Mininet assembling flow statistics instead of the purpose-built *cbench* on the *Forwarding Plane Host*, and a proactive loop for sending statistics requests implemented in the control application instead of only an event-driven handler promptly replying a fixed message on the *Controller Host*. This difference is shown by the reference Ethernet lines, as the RTT grows faster in Figure 4.3c. Nonetheless, in the 4G lines, it can be seen that it is no longer the bottleneck when packet sizes or concurrent exchanges are increased. The channel features are the dominating factor.

In summary, on a cellular network OpenFlow control plane, (i) the latency *lower bound is significant*, (ii) both the *packet size* and the number of *concurrent exchanges* can *increase the latency* as messages are buffered and the bandwidth occupation is reached, at which point (iii) the bandwidth *difference between the downstream and the upstream connection* proportionally affects the message RTTs.

The severity of the impact of the higher latency varies between different network functions, according to their reliance on the timeliness of the propagation of network events. Nevertheless, the measurements underline that, with the latency achieved nowadays, cellular networks can be seen only as a contingency mechanism for the control plane in scenarios where, otherwise, the network would be partitioned.

The latency penalty caused by high volume of control plane traffic is likely to occur in more reactive setups. For instance, a churn of unmatched flows triggering several *packet in* messages may not only fill the control plane channel bandwidth, but also overflow the forwarding device buffer. Then, the switch might send the whole packet embedded into the control plane message instead of an arbitrary number of header bytes. This can be prevented with particular configuration of the device behavior for table misses and installation of wild card rules, usually at the cost of conceding fine grain control.

The same idea applies to statistics collection. The volume of traffic can be reduced by the control application by decreasing the rate of requests, on a compromise with the timeliness of the network visibility.

5 EVALUATING THE FAILOVER TO THE BACKUP CELLULAR LINK

The evaluation of the RTT of OpenFlow message exchanges presented in the last chapter helps to characterize the behavior of a SDN control plane operating over a cellular network. However, the current state of technology and services indicate that this architecture is a better fit for a contingency resource, rather than for a primary control plane deployment, in particular because of the latency lower bound.

For a better characterization of the proposed backup mechanism, this chapter discusses the handover procedure from a primary wired control plane connection that fails to a backup cellular link. Guaranteeing a fast failover time on the control plane is crucial for minimizing the impact on forwarding plane operation. Three handover alternatives are described, and an experiment is conducted with each one of them in order to help answering the second question: **Q2.** *How quickly is the control plane handed over to the backup link when a failure occurs?*

5.1 Methodology

Using a cellular network device as a backup link implies defining a handover mechanism to be applied when a failure occurs in the primary wired link. The quality of such a mechanism is determined by the impact observed in the delivery of control plane messages during failure detection and recovery. OpenFlow control messages are exchanged over a classic TCP connection. Therefore, the natural way to accomplish the handover when a link failure partitions the subnetwork which the primary interface belongs to is to somehow detect that the connection is down and then to establish a new one using the backup interface.

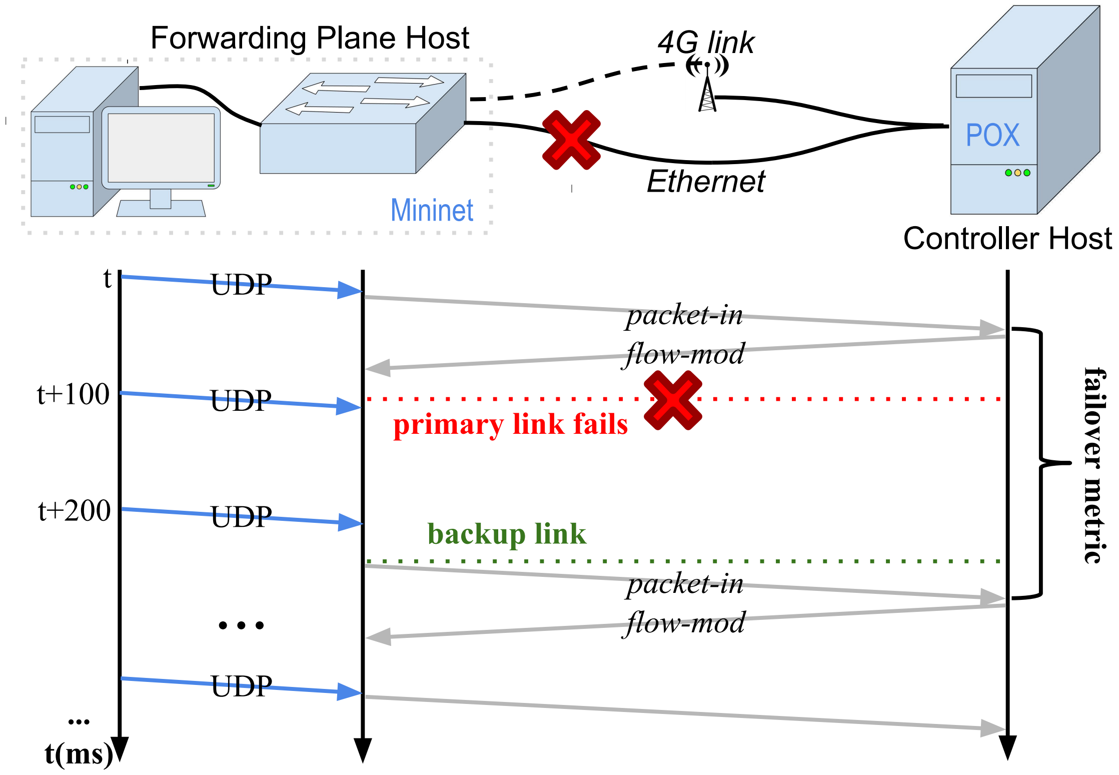
However, the time for detection and recovery can be smaller if the backup connection is set up earlier, before the failure event. The OpenFlow specification (OPEN NETWORKING FOUNDATION, 2013) provides for the configuration of backup controllers – i.e. on a different address and port – but not for backup paths to a same controller.

Since version 1.3, the OpenFlow protocol also presents a feature called *Auxiliary Connection*. However, this feature is not designed to provide a backup connection that takes over the control plane when the *Primary Connection* fails. It is rather intended to allow dividing tasks between parallel connections (e.g. rule installations and new flow notifications on the primary connection, and statistics collection on the auxiliary one). In fact, the protocol specifies that once a conforming switch detects that the main connection to a controller is broken, it must immediately close all its auxiliary connections to that same controller, enabling it to properly resolve *Datapath ID*[1] conflicts (OPEN NETWORKING FOUNDATION, 2013).

5.1.1 Experimental setup

The objective of the experiment is to measure the down time in control plane traffic after the failure on the primary wired link. As shown in Figure 5.1, Mininet is deployed with a single host connected to a forwarding device. The emulated host sends an UDP packet to the emulated switch every 100 ms to trigger *packet in* events, changing the destination port each time in order to cause different flow matches. The control application always responds with an actionless *flow mod* which exactly matches the packet flow, so that *packet in* messages keep being generated.

Figure 5.1: In this experiment, emulated host and switch run in Mininet at the *Forwarding Plane Host* and POX at the *Controller Host*. As before, there are two links between the hosts, and the Ethernet one is deactivated during communication to measure how long it takes for the control plane to recover.



Source: by author (2015).

[1]A *Datapath ID* uniquely identifies a datapath from the controller's point of view. It is usually composed from the switch MAC address and additional bits at the choice of the implementer.

The failover metric is the mean time between the last *packet in* received at the controller before the wired link is deactivated and the first one received afterwards, over 30 runs. Note that this metric is an upper bound to the actual failover time. There can be a small interval between the last message received and the failure, and another one just after the reestablishment of the channel and before the arrival of the next *packet in*.

First, the experiment is conducted on an unmodified Open vSwitch included in the Mininet image, to be presented in Subsection 5.1.2. As the analysis of the experiment will show that the majority of the down time is due to an implementation detail, the experiment is performed on a recompiled version of the software switch – as it will be described in Subsection 5.1.3 – removing a hard-coded lower threshold for the controller probe interval. Although the failover time is dramatically reduced, it is still limited by the probe intervals, as the procedure depends on detecting channel inactivity by timeout. Finally, in Subsection 5.1.4 we present an alternative approach that takes advantage of Multipath TCP, so that a same connection is maintained through disjoint paths[2] and the failure recovery can be accomplished quickly via the cellular link. While the latter option decreases the down time even further, it is subject to some extra requirements on both ends and on the underlying network.

5.1.2 Handover on Unmodified Open vSwitch

The *Forwarding Plane Host* is prepared for the failover by having gateways configured both for the wired and wireless media, with the *route* command. A lower metric is set on the gateway corresponding to the wired interface, to ensure that traffic will be primarily directed through it. A failure is inserted by taking down the Ethernet network interface of the *Forwarding Plane Host* during the loop of generation of UDP packets on the emulated host, forcing the new connection to be established through the gateway assigned to the surviving interface.

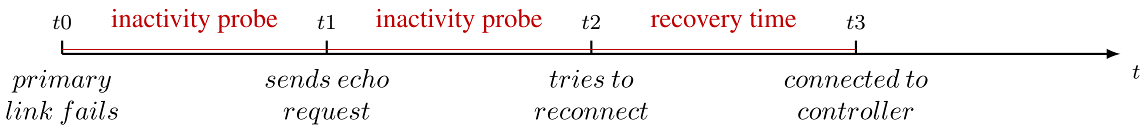
In this experiment, the link failure is always inserted in the first hop, for the sake of simplicity. Other failures along the path causing partition in the primary subnetwork could be detected on the TCP connection with an auxiliary mechanism, out of the scope of this work. For instance, an user space script could be configured to probe the controller with a *ping* command and then to alter the gateway setup with arbitrary periodicity and probe interval, establishing the control plane communication over the disjoint backup path.

[2]Throughout the text, this term does not necessarily mean an entirely disjoint path. It is enough to think that, at some point, the paths split into different subnetworks – namely, one of the paths goes through the cellular provider network.

The mean down time of the control plane was $12.487s$. A careful analysis of the experiment execution and of the Open vSwitch source code indicated that the dominating factor of this down time was the *inactivity probe* interval.

Open vSwitch allows the inactivity probe interval for the control plane channel to be configured. When the communication is idle for the configured amount of time, the switch probes the controller. If no answer is received after this same amount of time, it tries to reestablish the connection. However, the implementation enforces the probe interval to a minimum of 5 seconds, which is also the default value. This is true for Open vSwitch version 1.9.0, bundled in the Mininet 2.1.0, as well as up to its most recent build (by the time of writing), 2.3.0. The relevance of this feature motivated a repetition of the experiment with a modified software switch. The timeline in Figure 5.2 illustrates the failover process.

Figure 5.2: The time between $t0 - t1$ is the idle control plane time, after which the device probes the controller with an *echo request*. It waits for the same amount of time for an *echo reply* ($t1 - t2$). Finally, on $t2$ the switch tries to reach the controller through the backup link and the first *packet in* message arrives on $t3$.



Source: by author (2015).

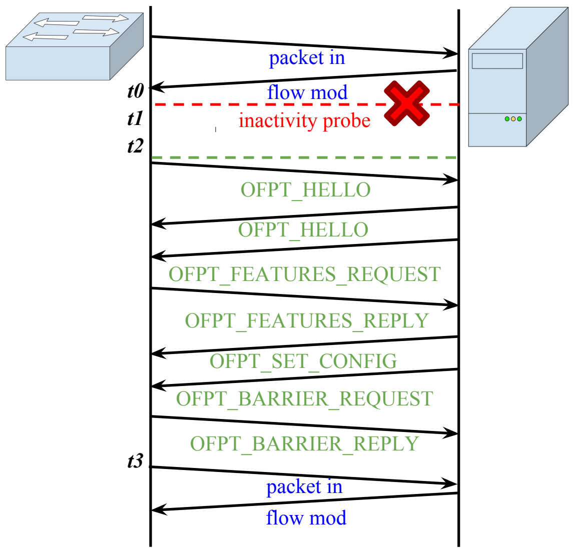
5.1.3 Handover on Open vSwitch with Lower Probe Interval

The 5-second lower threshold for the probe interval is enforced by two lines of code in the OpenFlow connection manager of Open vSwitch. Appendix B.3 shows the lines changed in the source files, bringing the limit down to 1 second.

Reducing the minimum inactivity probe interval increases the probability of false positives in detecting loss of the control plane connection. Nevertheless, this is a trade-off that can be determined by the network administration. This modification simply allows the inactivity probe interval to be set down to 1 second, but it does not prevent it from being configured to a higher value.

The failover timeline is the same from Figure 5.2. The probe interval is configured to the new lower threshold of 1 second, affecting both $t0 - t1$ and $t1 - t2$. The result is that the mean failover time dropped to $3.56s$. While the handover is lower-bounded to 10 seconds – i.e. twice the inactivity probe interval – with the original Open vSwitch, the small modification described here brings down this limit to 2 seconds. In fact, the handover time exceeding this lower bound is the sum of the time spent in the OpenFlow connection procedure and the interval from completing the connection establishment to the first *packet in* arrival. The basic handshake which takes place at $t2 - t3$ is illustrated at Figure 5.3.

Figure 5.3: When the primary connection is lost and the inactivity probe interval expires ($t0 - t2$), a new OpenFlow conenction is established using the backup gateway. A switch starts the connection by sending a *hello* message, to which the controller replies. This exchange negotiates the protocol version to be used in the channel. The controller then queries the switch about information such as the Datapath ID, its capabilities and number of tables, with the *features request and reply* exchange. Next, the controller configures switch parameters, such as the number of bytes to be embedded in *packet in* messages, with *set config*. Finally, a *barrier* message exchange completes the handshake and separates the connection procedure from normal operation ($t3$).



Source: by author (2015).

5.1.4 Handover with Multipath TCP

Although the recovery time is greatly improved by reducing the inactivity probe interval, it is still limited by a fixed lower bound of twice the configured time plus the initial handshake of Figure 5.3 when reconnecting. A more sophisticated way to perform the handover is to create the backup connection in advance. However, as described earlier, the OpenFlow provision for backup controllers supposes that the multiple connections are each to a different controller. In fact, the protocol specification stresses that the coordination among control instances is on their own behalf, and describes a set of roles – e.g. *Master*, *Slave*, *Equal* – to which the instances must assign themselves.

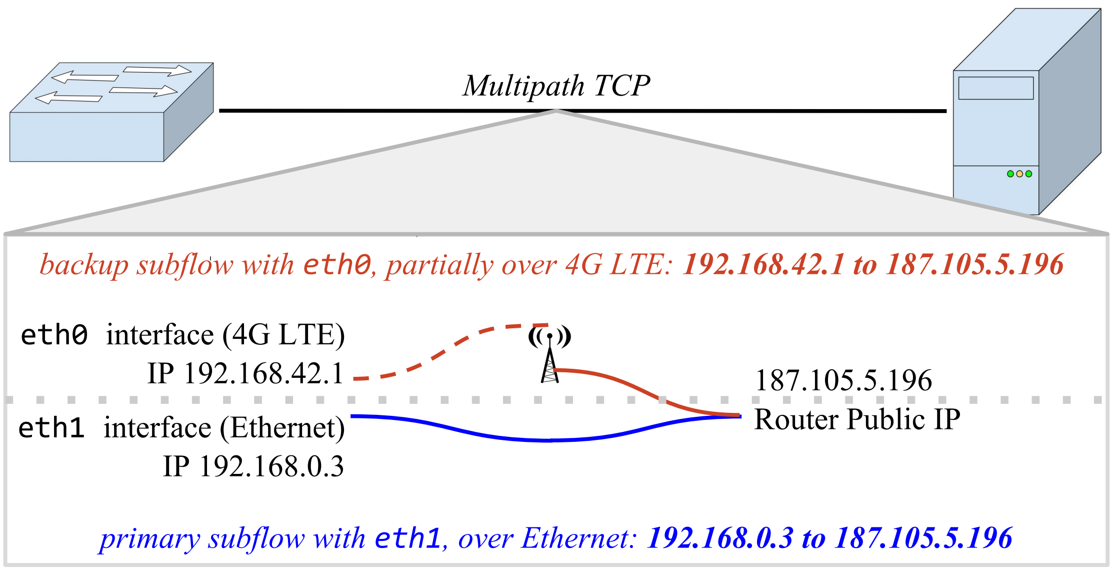
One can think of a scenario where a single controller is multihomed, with each of its interfaces on a separate network, providing disjoint paths for a switch to establish its control plane channels upon. Therefore, a switch could be configured to have *Master* and *Slave* role controllers assigned to the addresses corresponding to each one of the multihomed interfaces. Ideally, this would provide for a shorter handover time, as the backup connection would be already established before an occasional failure.

There are two obstacles to this approach, though. First, both addresses of the controller would have to be accessible for incoming connections. As mentioned before, mobile clients are usually behind NATs, and this is the case for the cellular link in our testbed. Second, and more importantly, the controller would have to be able to identify that the parallel connections were intended to provide redundancy. Controllers usually rely on a unique *Datapath ID* that identifies a switch on incoming control plane connections. This is useful for maintaining state in the case of a temporary connection failure. While using the same ID on both connections could allow the opposite end to identify a given switch, it would also mean that this ID would no longer be unique, and that the controller's connection manager would have to drop that assumption. For instance, in the event of a second connection for a given *Datapath ID* arriving, the first one should not be dropped as obsolete.

However, a similar approach can provide the early establishment of the backup connection on a disjoint path through a multihomed environment by leveraging a recent development placed at the Transport layer. With Multipath TCP (PAASCH; BARRE et al., 2015) installed at both ends, separate connections – called *subflows* – can be created for any pair formed by one IP address from each of the two ends of the path. As the *subflows* are transparent to the upper layer, the multihomed host can be on the switch end or, as usually referred to in our testbed, the *Forwarding Plane Host*. Either end of the connection can start new *subflows* and advertise alternative IP addresses to the opposite end. Therefore, the mechanism can operate even with the cellular interface behind a NAT.

The implementation of Multipath TCP is described in RFC 6824 (FORD et al., 2013). The protocol identifies multiple paths by the presence of multiple addresses at hosts. As in the previous handover experiments, both *Forwarding Plane Host* interfaces have their gateways configured, only with the cellular interface being signaled as a backup path to the MPTCP module. Therefore, when the control plane multipath connection is established, two *subflows* are initiated – one for each path, as illustrated in Figure 5.4 – but traffic is not directed through the wireless interface, except in the event of a failure that causes it to be the only remaining active *subflow*. This is to avoid unnecessary extra latency due to a share of the control plane packets being sent via the cellular network, as Multipath TCP allows traffic to be balanced among available paths.

Figure 5.4: Despite the fact that both ends of the path must have a kernel with MPTCP support in the transport layer, the multipath connection is transparent to the upper layers. Both the switch and the controller are oblivious to the multiple paths. Within the connection, the cellular link is configured as a backup interface. The wireless *subflow* is established, but receives no traffic unless the other *subflow* fails.


Multipath TCP
backup subflow with eth0, partially over 4G LTE: 192.168.42.1 to 187.105.5.196
eth0 interface (4G LTE)
IP 192.168.42.1
187.105.5.196
Router Public IP
eth1 interface (Ethernet)
IP 192.168.0.3
primary subflow with eth1, over Ethernet: 192.168.0.3 to 187.105.5.196

```
Backup interface configuration with Multipath TCP:
> ip link set dev eth0 multipath backup
```

Source: by author (2015).

The use of this mechanism is very promising, as shown by the mean failover time being brought down to $1.584s$. However, it is still under development, with the RFC in *Experimental* status. Additionally, there are some restrictions on the behavior of middleboxes along the path. The signaling for multipath management is performed with an optional TCP header field. Middleboxes should ideally forward packets without changes to the TCP options. However, it is known that some elements such as NATs, PEPs, traffic normalizers, firewalls or IDSs can behave differently, either by stripping options, dropping packets with new options, replicating an option when doing segmentation or dropping options when coalescing segments (FORD et al., 2013). Since the protocol is designed to fall back to single-path TCP behavior when facing these issues, the primary control plane operation would be maintained when passing through these middleboxes, but the handover mechanism would be lost.

5.2 Experimental Results

Table 5.1: Failover process results (s)

	Description	Minimum	Average	Maximum	Std. Dev.
5.1.2	Open vSwitch 5 seconds probe	11.491	12.487	14.864	0.865
5.1.3	Open vSwitch 1 second probe	2.600	3.560	8.673	1.434
5.1.4	Open vSwitch with MPTCP	1.017	1.584	3.871	0.517

Source: by author (2015).

Table 5.1 compares the experiments by showing the minimum, mean, maximum time and standard deviation between the last received communication from the OpenFlow switch and the first *packet in* message after recovery in (a) an unmodified Open vSwitch 1.9.0 with a 5 second inactivity probe interval, (b) in a 1 second probe deployment and (c) in an unmodified switch on a Multipath TCP environment.

In summary, the experiment results show that the *failover time is dominated by probe interval on both single-path TCP scenarios*. The handover is possible on an original Open vSwitch, but the down time can be greatly reduced if the minimum inactivity probe interval is lowered. No changes are needed in the controller, and there are no additional requirements on the underlying network elements. However, it depends on some additional failure detection mechanism to trigger the change of gateway. Likewise, the inverse procedure should be performed when the connectivity is restored in the primary control plane channel, which is not covered in this work.

Multipath TCP allows for a quicker recovery, as the OpenFlow connection is not interrupted and there is no need to perform a new handshake. Although the controller application software can be oblivious to it, the underlying kernel in both ends – switch and controller – must implement Multipath TCP. The additional module might incur in additional overheads in the switch, in particular if the device has a modest processing unit. Additionally, for the handover to be available, there are some requirements on the behavior of middleboxes towards TCP option headers.

6 EVALUATING A NETWORK FUNCTION ON A CELLULAR CONTROL PLANE

One of the most attractive features of OpenFlow is the ability to perform decisions based on a global view of the network, simplifying network programmability and consequently the deployment of network policies. The proposed architecture, enhanced with wireless links, is an effort towards the maintenance of network visibility within the controller. Even if the control plane partition can be prevented and the global view maintained with this architecture, some performance limitations have been shown in Chapter 4, and they might have an impact on the operation of network functions. This leads to the third question: **Q3.** *How well do network functions that rely on a snapshot of the network state behave on such an architecture?*

After obtaining a baseline of the cellular channel behavior and experimenting with the control plane handover procedure from the wired primary to the wireless backup link, this chapter evaluates a load balancing application that relies on network state awareness operating on the proposed architecture.

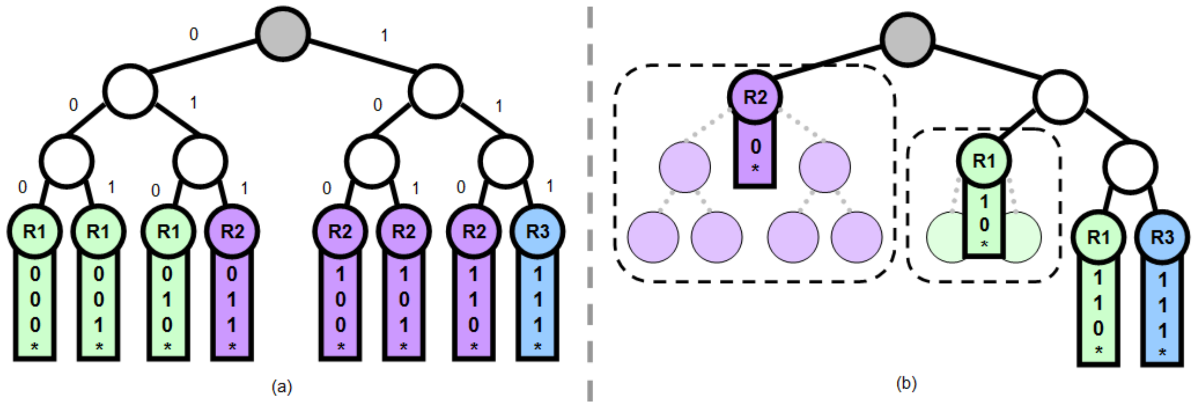
6.1 Methodology

The third and final question attempts to verify whether network applications which depend upon global visibility can still perform well under the conditions imposed by this alternative medium. Load balancing is usually employed in the literature as an example of network function that can leverage the benefits of SDN, such as global visibility (LEVIN et al., 2012; SCHULZ-ZANDER; SARRAR; SCHMID, 2014), flexible switches replacing some of the purpose-built hardware (KOERNER; KAO, 2012; GANDHI et al., 2014), grain control of rules (WANG; BUTNARIU; REXFORD, 2011; CURTIS et al., 2011; BREDEL et al., 2014) and statistics collection (LI; PAN, 2013). Even in solutions that argue for returning some of the functionality to the forwarding plane, such as employing network programmability and visibility with packet instrumentation (JEYAKUMAR et al., 2014), load balancing is one of the use cases. The scheme for balancing will be described at Subsection 6.1.1, followed by the experimental setup for traffic generation and measurements on Subsection 6.1.2.

6.1.1 Network Function: Load Balancing

In this experiment, an address prefix load balancing scheme inspired in the work of Wang *et al.* (WANG; BUTNARIU; REXFORD, 2011) is evaluated comparatively in both control plane media. A binary tree for partitioning the space of source IP addresses among different server instances is built reactively, according to the latest readings on the usage of switch ports, and wildcarded rules are installed in the flow table for the forwarding plane traffic.

Figure 6.1: IP prefix-based binary tree. Server replicas are *R1*, *R2* and *R3*, with assigned weights $\alpha[1] = 3$, $\alpha[2] = 4$ and $\alpha[3] = 1$. In (a), leaf nodes are mapped to a total of 8 flow ranges. But the number of wildcard rules can be smaller for the same weight assignment (b).



Source: Wang, Butnariu and Rexford (2011).

A fully reactive scheme in which the controller interferes at each incoming flow would certainly suffer a greater performance impact when operating over a cellular network control plane. However, such a scheme is usually considered impracticable in terms of throughput and latency (GANDHI et al., 2014), due to the overhead of processing such a fine grain fully on software. As illustrated by related work cited at the start of this section, the trend in the area is to limit as much as possible the use of a reactive scheme in the software-defined part of the system. For example, this is achieved with proactive installation of rules, or with hybrid approaches that manage the trade-off between the performance of a hardware load balancer and the flexibility of a programmable switch controlled by an application with global visibility.

Conversely, a fully proactive rule installation scheme would hide most of the effects of the control plane latency, except for some delay in the controller's view of the network state. In the strategy chosen for this experiment, rules are still installed reactively in the first occurrence of a flow. However, the new rule is set to match a range of IP addresses adjacent to the source address of the incoming packet. Any traffic from this range arriving during the live time of the rule will be matched and forwarded without control plane intervention. Additionally, each rule has an *idle timeout* of $1s$. If no packet matching the range of a rule arrives during this interval, the switch automatically removes the rule from its flow table. The next time a packet from that range reaches the switch, a new reactive rule installation will take place. In contrast to a fully reactive scheme, this strategy does not concentrate all the consequences of control plane latency on forwarding delay, but rather shares this impact with the effectiveness of the load balancing – for instance, flow rules based on stale state are more susceptible to perform unequal balancing.

Figure 6.1 illustrates the binary tree scheme proposed by Wang *et al.* (WANG; BUTNARIU; REXFORD, 2011). One of the objectives of the authors was to keep the number of flow rules to be installed as low as possible. In the example, the IP address space is divided into 8 flow ranges, each identified by a prefix of 3 bits. Three server replicas have weights assigned to them, indicating their proportional share of address ranges to receive traffic from. Two possible mappings are shown between flow ranges and server replicas. The left hand side mapping is sub-optimal in terms of minimizing the number of rules, even if some of the rules could be merged – for instance, $000*$ with $001*$ becoming $00*$ towards *R1*, and $100*$ with $101*$ as $10*$ to *R2*. In contrast, the right hand side mapping assigns the server replicas to a set of flow ranges that takes full advantage of wildcard rules. In fact, the reason for grouping IP addresses by prefix is an OpenFlow limitation that restricts the use of wildcards in the lower order bits of an address match. The authors claim that, ideally, the mapping should be based on the suffix, as those bits have greater entropy, but they settled with the IP prefix as a workaround.

In the experiment performed in this dissertation, the server weights are reassigned periodically, according to the traffic directed to the replica in the latest time slot. Let f be the number of flow ranges in which the IP address space is divided. The number of server replicas to which incoming traffic is balanced is given by s. The number of bytes transferred in a server replica $r, 1 \leq r \leq s$ during time slot t is represented as $b_t[r]$. Let c_t be a normalization coefficient obtained from the flow ranges term f divided by the sum of the inverses of the traffic passed through all servers:

$$c_t = \frac{f}{\sum\limits_{i=1}^{s} b_t[i]^{-1}} \tag{6.1}$$

The weight $\alpha_t[r]$ assigned to replica $r, 1 \leq r \leq s$ in time slot t is given by:

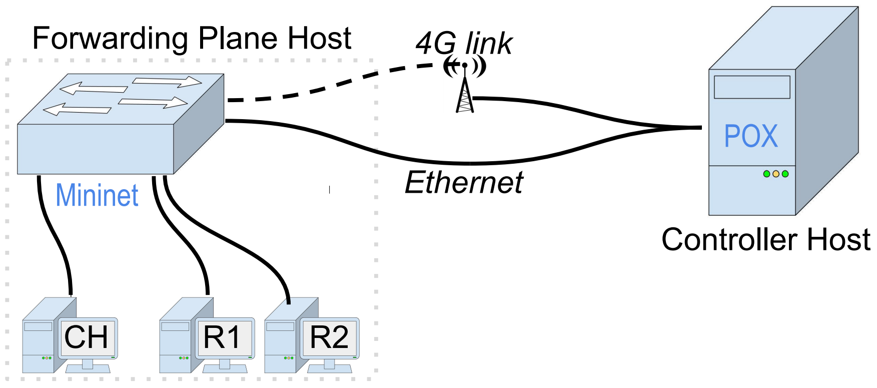
$$\alpha_t[r] = \left\lfloor \frac{c_{t-1}}{b_{t-1}[r]} \right\rfloor \tag{6.2}$$

The weight assigned to a server for a given time slot is the floor of the division of the normalization coefficient by the number of bytes directed to it during the latest time slot. Basically, server weights are the inverses of the relative traffic directed through them in the previous interval. The weight is truncated towards zero to become an integer value so as to obtain a direct mapping between a server's weight and the number of flow ranges assigned to it. When the floor function causes the sum of the integer weights to be less than f, the single server with the top weight $\alpha_t[\arg\max_r \alpha_t[r]]$ is incremented by $f - \sum_{i=1}^{s} \alpha_t[i]$.

6.1.2 Experimental Setup

Figure 6.2 shows three emulated hosts connected to the emulated switch. One host produces traffic directed to a given destination IP address, which must be balanced among the server replicas.

Figure 6.2: CH (*client host*) produces traffic with different IP sources towards either emulated host *server replicas* ($R1$ or $R2$), passing through the emulated switch. Routing rules are configured according to the load balancer running at POX at the *Controller Host*.



Source: by author (2015).

A total of 6000 UDP packets are sent, scheduled according to a Poisson process with a mean interval of $10ms$ between consecutive packets. The size of each packet is exponentially distributed with a mean of 1024 bytes, upper-bounded by the Ethernet frame payload limit (1500 bytes including IP and UDP headers). The source IP is spoofed from a range of 1024 sequential addresses, simulating the same number of distinct clients, in order to allow the range-based load balancing to take place. For all conducted experiments, the generated load is the same. Each incoming packet incurs either in a *table miss* on the switch, causing a rule to be reactively installed by the controller, or in a *rule match*. Traffic is directed towards two server replicas, according to matching rules defined by the load balancing application policy.

The set of parameters on the control application configures the load balancer *timeliness* features. This refers to the speed of reaction of the controller to a change in state, building an appropriate matching between addresses and servers and being less prone to perform unequal balancing. The controller periodically sends *stats requests* to the OpenFlow switch, becoming aware of the current state of the servers corresponding ports. At each *stats reply* arrived, the server weights are recalculated according to Equation (6.2) and the tree is adjusted. The interval between each statistics request message is varied between $0.5s$, $1s$ and $2s$. A longer interval decreases control traffic at the cost of possibly reacting too late, due to stale state. Timeliness is particularly configured by adjusting the granularity of the rules. Since the generated load is repeated, the granularity is set by splitting the source IP address space of the experiment (1024) into 32, 64 and 128 flow ranges. For each input value, Table 6.1 shows the minimum rule granularity (i.e. number of IP addresses matched by a single rule) and the range of rules installed when using an optimal tree mapping for two servers.

Table 6.1: The top row shows the number of flow ranges in which the address space is subdivided. The minimum rule granularity defines the number of addresses matched by a rule assigned to a single flow range. The optimal number of rules to be installed is minimal when both weights are the same and maximal when the top weight equals $f - 1$.

Flow ranges	f	32	64	128
Mininum rule granularity	$1024/f$	32	16	8
Optimal number of rules with 2 servers	$[2 \ldots 1 + \log_2 f]$	$[2 \ldots 6]$	$[2 \ldots 7]$	$[2 \ldots 8]$

Source: by author (2015).

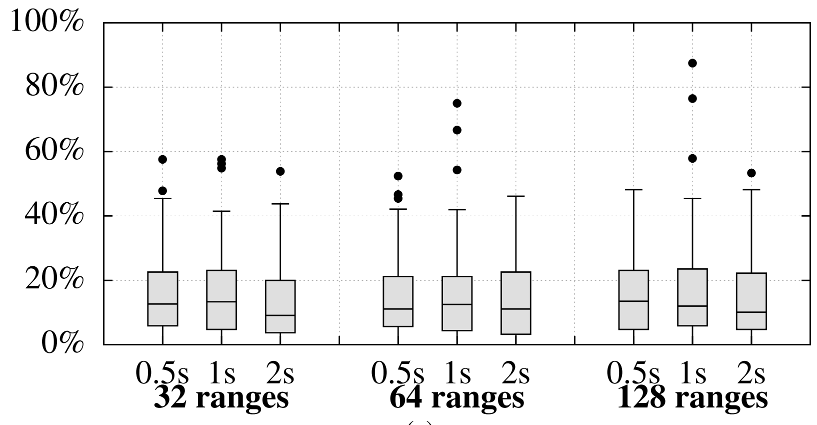
A more *timely* controller should perform more efficient load balancing at the cost of higher control link occupancy. The objective of this experiment is to evaluate the impact of using a cellular control link over forwarding plane services, in particular due to the added delay in receiving port statistics reports and in the reactive installation of rules. The negative impact to load balancing is measured by comparing the relative usage of the server ports. Measurements of port usage are taken at the *Forwarding Plane Host* using Open vSwitch *ovs-ofctl dump-ports* interface, since the controller readings are dependent on the periodicity of statistics collection – which is one of the input parameters – and are part of the object of evaluation.

Additionally, the mean forwarding plane throughput is obtained to assess whether the control reaction time for forwarding decisions causes a significant delay in packet delivery. The statistics request interval is fixed at $1s$. The address space is divided into 32, 64, 128, 256 and 512 flow ranges. A greater number of flow ranges tends to decrease the rule granularity (i.e. each rule matches a smaller number of addresses). Given that each rule is installed reactively, the higher the number of flow ranges, the higher the probability of flow table misses.

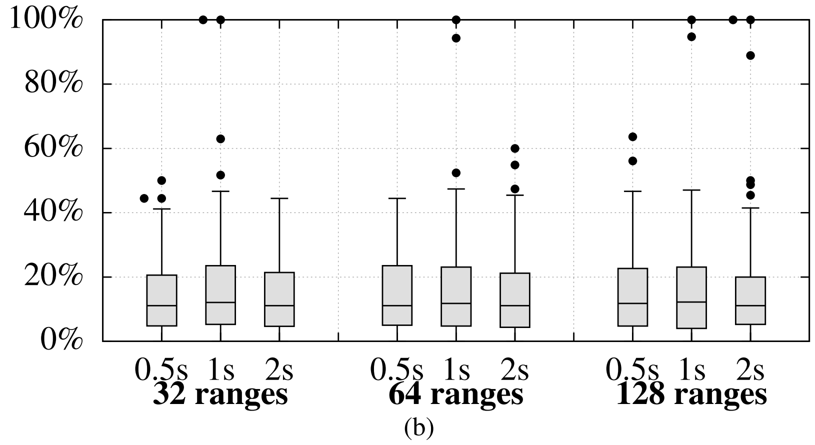
6.2 Experimental Results

In Figure 6.3, the results for relative usage of server ports along the experiment duration are presented in box plots. For instance, a value of 0% means that the load is equally balanced among the two servers. 100% stands for a case where the load is completely forwarded to one of the servers. The usage is measured in intervals of $0.5s$.

Figure 6.3: Relative usage of two server ports with a control plane link over (6.3a) Ethernet and (6.3b) 4G. In each plot, the number of IP address subranges and the periodicity of port status querying by the controller are varied to change the *timeliness* of the load balancing function.



(a)



(b)

Source: by author (2015).

In summary, *load is balanced seamlessly and, on average, efficiently*. The higher latency in the 4G control plane links has caused occasional late reactions in the load balancing process, as illustrated by the outliers in Figure 6.3b. The experiments conducted with an Ethernet control link feature a more consistent behavior (Figure 6.3a). Nevertheless, the average load balancing effectiveness is very similar in both cases. Since the variation in the number of flow sub-ranges and in the periodicity of statistics collection shows little significance in the measured results, it can be observed that the major drawback of using 4G links in the control plane for this load balancing function is the occasional mistaken forwarding decision due to a delayed view of the forwarding plane state.

Table 6.2: Effect of control latency on data throughput. Statistics request interval is fixed at $1s$.

Flow ranges		32	64	128	256	512
Ethernet	kb/s	587.01	593.98	584.46	568.98	590.72
	Rules	156	880	2211	3700	4650
4G	kb/s	593.44	574.25	596.31	588.68	588.55
	Rules	224	1078	2328	3719	4624

Source: by author (2015).

An analysis of the throughput measurements indicates that the load balancing application has little or no impact in the timeliness of forwarding plane packet delivery. Table 6.2 shows that, even with much higher rate of flow installations due to smaller address sub-ranges, the resulting throughput is consistent among all measurements in both media.

It is worthy of note that this rather favorable behavior is due in part to two factors: the limited size of the testbed and the policy chosen for the installation of flow rules. Topologies were kept simple in order to avoid additional overheads from the emulated environment. During the development of the evaluation of our paper (PETRY; SILVA; BARCELLOS, 2015) and this dissertation, we observed that the shared CPU time between the hosts and switches within Mininet had a significant impact in the results. This is the reason why the generated traffic was kept within a single host with IP source address spoofing, rather than a set of hosts with distinct assigned IP addresses. The set of servers for which the load is balanced was also kept on a minimal size. In spite of the fact that the chosen policy features a reactive installation of flow rules, each one of them matches a set of addresses. The coverage of the rule match varies according to both the flow ranges parameter and the mapping between addresses and servers, performed with the binary tree scheme. Therefore, in practice, some of the generated packets find a matching rule that was installed proactively, even if it is the first time that a packet with that source address is generated.

The latency lower bound obtained with current 4G LTE technology is still significantly high, whether it is around $120ms$ as measured in the experiments of Chapter 4 or below $100ms$ as documented in the literature in other regions of the world (CHEN et al., 2013). For network applications designed to rely heavily on reactive installation of rules, the use of the backup links in the control plane imposes a significant performance drawback on the forwarding plane due to the latency lower bound.

Fortunately, the use of an appropriate load balancing scheme allows to avoid most of the impact of the control plane latency over the forwarding plane latency and throughput, rather trading it for occasionally losing efficiency in the load balancing. The ability of adapting a given network function to the higher control plane latency defines the suitability of this backup architecture for that network function.

7 CONCLUSION

To the best of the author's knowledge, this work features the first analysis of the operation of OpenFlow control traffic over cellular links, that being the main contribution of this dissertation. The experimental study provides insights towards answering the questions regarding the effects of cellular links in OpenFlow traffic, the failover procedure to the backup links and the operation of a network function with a cellular control plane channel.

Among the insights of the analysis, it is worth noting that the experiments show evidence that *backup cellular links on the control plane allow an OpenFlow-based network to maintain operation of network functions under certain circumstances*, in spite of providing lower bandwidth and higher latency than wired links. Nevertheless, this architecture is not currently suitable for completely replacing an out-of-band wired control plane network. Instead, it can be seen as a simple, non-expensive backup control architecture for maintaining a consistent global network state view and limited reactiveness for control plane decisions. With proper configuration and the help of handover technologies like Multipath TCP, this backup architecture is able to quickly come to action when failures happen in the primary links.

The suitability of the resilient architecture proposed in this dissertation *is closely related to the requirements of the network functions and to the way that control applications are programmed.* As a rule of thumb, the more a network function blocks on events that depend on controller intervention, the more this function is affected by the latency of a cellular network control plane. Additionally, the effect can be worsened by control traffic churn, due to a large number of concurrent events.

In comparison to related mechanisms that attempt to tackle control plane partitions, *this architecture is intended to maintain a consistent, centralized, global visibility of the network, even if somewhat delayed.* Distributed controller instances often feature a trade-off between the timeliness of state and such global visibility, which is also prone to delays due to consensus algorithms. When favoring timeliness, techniques usually rely on locality of operations, which is also subject to the way control applications are programmed. Forwarding plane mechanisms in general are designed to accomplish tasks without relying on global visibility, and are therefore complementary to the other approaches. Nevertheless, this implies separating some of the network functionality from the logically centralized behavior.

As future work is concerned, applying the concept of wireless links for communication between physically distributed modules of logically centralized controllers or for interconnecting different controller instances could be explored. While the former alternative should evolve in the pace that new flavors of open-source distributed controllers become available and novel control plane distribution insights are discovered, the latter would depend on the convergence of an east/westbound API standard.

Another possible line of work is to explore the cellular link placement in a control plane topology. This could include sharing a backup cellular link for the control channel of several nearby switches, considering a trade-off between the link occupation by shared control traffic and the cost of installation and services. The architecture proposed and evaluated here has cellular interfaces attached to the switch end of the connection, due to the restrictions of the available testbed and reflecting the most common scenario on mobile networks. However, mobile service providers in some regions of the world do offer public reachable interfaces for an additional fee, which in turn could be used at the controller end as a backup interface for the control plane. This option tends to be strengthened as IPv6 networks spread.

The natural extension to the evaluation of the effects of a cellular link on OpenFlow traffic would be to waive the assumptions made on signal quality. In particular, this would include performing a long-term experiment (e.g. covering all periods of the day, comparing different week days, during several months, with different mobile network operators) in an attempt to identify patterns of variation in the signal quality and characterize its effects. Another possible extension is to augment the experiment, for instance, by exploring features of alternative control plane evaluation tools (such as OFCProbe (JARSCHEL et al., 2014)) or by obtaining OpenFlow traces of production environments.

The handover experiments were based on Open vSwitch, which is a popular choice as the base system of OpenFlow forwarding devices. The gateway configurations for multihoming and the use of Multipath TCP could be generalized to other switches inasmuch as their corresponding implementations either natively support these mechanisms or allow the proposed configurations or modifications. Multipath TCP connections share the packet delivery guarantees of single-path TCP, given that at least one of the subflows remains active. While this assures there are no packet losses on the control plane, a deeper assessment on the impact of a handover on forwarding plane operation would be an interesting extension to this experiment.

Load balancing is frequently chosen as an use case of application based on global network view. It allows for a wide space of schemes and configurations that result in different controller reactivity requirements. Notwithstanding, the evaluation of a cellular link control plane could be extended to include more complex network functions, such as IDSs or dynamic policy-based routing. The first challenge would be to identify how does staleness of state affect the functions in question.

In general, the experiments would benefit from a testbed more resembling to production networks, for instance, using larger topologies deployed outside of an emulation environment and featuring multiple OpenFlow switches, including retail (not only virtual) devices.

REFERENCES

AKELLA, A.; KRISHNAMURTHY, A. A Highly Available Software Defined Fabric. In: 13TH ACM WORKSHOP ON HOT TOPICS IN NETWORKS, 2014, Los Angeles, CA, USA. **Proceedings...** New York, NY, USA: ACM, 2014. (HotNets-XIII), p. 21:1–21:7.

BARI, M. F. et al. Dynamic Controller Provisioning in Software Defined Networks. In: 9TH INTERNATIONAL CONFERENCE ON NETWORK AND SERVICE MANAGEMENT, CNSM, 2013, Zürich, Switzerland. **Proceedings...** Zürich, Switzerland: IEEE, 2013. p. 18–25.

BEHESHTI, N.; ZHANG, Y. Fast failover for control traffic in Software-defined Networks. In: IEEE GLOBAL COMMUNICATIONS CONFERENCE (GLOBECOM), 2012, Anaheim, CA, USA. **Proceedings...** Anaheim, CA, USA: IEEE, 2012. p. 2665–2670.

BERDE, P. et al. ONOS: Towards an Open, Distributed SDN OS. In: THIRD WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, 2014, Chicago, Illinois, USA. **Proceedings...** New York, NY, USA: ACM, 2014. (HotSDN ’14), p. 1–6.

BREDEL, M. et al. Flow-based load balancing in multipathed layer-2 networks using OpenFlow and multipath-TCP. In: THIRD WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, 2014, Chicago, Illinois, USA. **Proceedings...** New York, NY, USA: ACM, 2014. (HotSDN ’14), p. 213–214.

CANINI, M. et al. A Distributed and Robust SDN Control Plane for Transactional Network Updates. In: IEEE INFOCOM 2015 - IEEE CONFERENCE ON COMPUTER COMMUNICATIONS, 2015, Hong Kong, China. **Proceedings...** Hong Kong, China: IEEE, 2015.

CHEN, Y.-C. et al. A Measurement-based Study of MultiPath TCP Performance over Wireless Networks. In: 2013 CONFERENCE ON INTERNET MEASUREMENT CONFERENCE, 2013, Barcelona, Spain. **Proceedings...** New York, NY, USA: ACM, 2013. (IMC ’13), p. 455–468.

CHOWDHURY, S. et al. PayLess: A low cost network monitoring framework for Software Defined Networks. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2014, Krakow, Poland. **Proceedings...** Krakow, Poland: IEEE, 2014. p. 1–9.

CURTIS, A. R. et al. DevoFlow: Scaling Flow Management for High-performance Networks. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 41, n. 4, p. 254–265, aug. 2011. ISSN 0146-4833.

DIXIT, A. et al. ElastiCon: an elastic distributed sdn controller. In: TENTH ACM/IEEE SYMPOSIUM ON ARCHITECTURES FOR NETWORKING AND COMMUNICATIONS SYSTEMS, 2014, Los Angeles, California, USA. **Proceedings...** New York, NY, USA: ACM, 2014. (ANCS ’14), p. 17–28.

FERGUSON, A. D. et al. Participatory networking: An API for application control of SDNs. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 43, n. 4, p. 327–338, aug. 2013. ISSN 0146-4833.

FORD, A. et al. TCP Extensions for Multipath Operation with Multiple Addresses (RFC 6824), **IETF Request for Comments**. Internet Engineering Task Force, 2013. Accessed: Apr 18 2015. Available from Internet: <http://tools.ietf.org/html/rfc6824>.

GANDHI, R. et al. Duet: Cloud Scale Load Balancing with Hardware and Software. In: 2014 ACM CONFERENCE ON SIGCOMM, 2014, Chicago, Illinois, USA. **Proceedings...** New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 27–38.

GUDE, N. et al. NOX: Towards an Operating System for Networks. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 38, n. 3, p. 105–110, jul. 2008. ISSN 0146-4833.

GUIMARAES, C.; CORUJO, D.; AGUIAR, R. L. Enhancing openflow with Media Independent Management capabilities. In: COMMUNICATIONS (ICC), 2014 IEEE INTERNATIONAL CONFERENCE ON, 2014, Sydney, Australia. **Proceedings...** Sydney, Australia: IEEE, 2014. p. 2995–3000.

HASSAS YEGANEH, S.; GANJALI, Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In: FIRST WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKS, 2012, Helsinki, Finland. **Proceedings...** New York, NY, USA: ACM, 2012. (HotSDN '12), p. 19–24.

HASSAS YEGANEH, S.; GANJALI, Y. Beehive: Towards a Simple Abstraction for Scalable Software-Defined Networking. In: 13TH ACM WORKSHOP ON HOT TOPICS IN NETWORKS, 2014, Los Angeles, CA, USA. **Proceedings...** New York, NY, USA: ACM, 2014. (HotNets-XIII), p. 13:1–13:7.

HELLER, B.; SHERWOOD, R.; MCKEOWN, N. The Controller Placement Problem. In: FIRST WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKS, 2012, Helsinki, Finland. **Proceedings...** New York, NY, USA: ACM, 2012. (HotSDN '12), p. 7–12.

HEWLETT-PACKARD DEVELOPMENT GROUP. **HP Virtual Application Networks SDN Controller**. 2013. Accessed: Mar 30 2015. Available from Internet: <http://h17007.www1.hp.com/docs/networking/solutions/sdn/4AA4-8807ENW.pdf>.

HU, Y. et al. Reliability-aware controller placement for Software-Defined Networks. In: INTEGRATED NETWORK MANAGEMENT (IM), 2013 IFIP/IEEE INTERNATIONAL SYMPOSIUM ON, 2013, Ghent, Belgium. **Proceedings...** Ghent, Belgium: IEEE, 2013. p. 672–675.

ISOLANI, P. H. et al. Interactive monitoring, visualization, and configuration of OpenFlow-based SDN. In: INTEGRATED NETWORK MANAGEMENT (IM), 2015 IFIP/IEEE INTERNATIONAL SYMPOSIUM ON, 2015, Ottawa, Canada. **Proceedings...** Ottawa, Canada: IEEE, 2015.

JAIN, S. et al. B4: Experience with a Globally-deployed Software Defined WAN. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 43, n. 4, p. 3–14, aug. 2013. ISSN 0146-4833.

JARSCHEL, M. et al. OFCProbe: A platform-independent tool for OpenFlow controller analysis. In: COMMUNICATIONS AND ELECTRONICS (ICCE), 2014 IEEE FIFTH INTERNATIONAL CONFERENCE ON, 2014, Da Nang, Vietnam. **Proceedings...** Da Nang, Vietnam: IEEE, 2014. p. 182–187.

JEYAKUMAR, V. et al. Millions of Little Minions: Using Packets for Low Latency Network Programming and Visibility. In: 2014 ACM CONFERENCE ON SIGCOMM, 2014, Chicago, Illinois, USA. **Proceedings...** New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 3–14.

KOERNER, M.; KAO, O. Multiple service load-balancing with OpenFlow. In: HIGH PERFORMANCE SWITCHING AND ROUTING (HPSR), 2012 IEEE 13TH INTERNATIONAL CONFERENCE ON, 2012, Belgrade, Serbia. **Proceedings...** Belgrade, Serbia: IEEE, 2012. p. 210–214.

KOPONEN, T. et al. Onix: A Distributed Control Platform for Large-scale Production Networks. In: 9TH USENIX CONFERENCE ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 2010, Vancouver, BC, Canada. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2010. (OSDI'10), p. 1–6.

KREUTZ, D. et al. Software-Defined Networking: A Comprehensive Survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In: 9TH ACM SIGCOMM WORKSHOP ON HOT TOPICS IN NETWORKS, 2010, Monterey, California. **Proceedings...** New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6.

LEVIN, D. et al. Logically Centralized?: State Distribution Trade-offs in Software Defined Networks. In: FIRST WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKS, 2012, Helsinki, Finland. **Proceedings...** New York, NY, USA: ACM, 2012. (HotSDN '12), p. 1–6.

LI, L. E.; MAO, Z. M.; REXFORD, J. Toward Software-Defined Cellular Networks. In: SOFTWARE DEFINED NETWORKING (EWSDN), 2012 EUROPEAN WORKSHOP ON, 2012, Darmstadt, Germany. **Proceedings...** Darmstadt, Germany: IEEE, 2012. p. 7–12.

LI, Y.; PAN, D. OpenFlow based load balancing for Fat-Tree networks with multipath support. In: COMMUNICATIONS (ICC), 2013 IEEE INTERNATIONAL CONFERENCE ON. **ICC 2013**. 2013. p. 1–5. Accessed: Apr 30 2015. Available from Internet: <http://users.cis.fiu.edu/~pand/publications/13icc-yu.pdf>.

LIU, J. et al. Ensuring Connectivity via Data Plane Mechanisms. In: 10TH USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION, 2013, Lombard, IL. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2013. (nsdi'13), p. 113–126.

LTE. **LTE - The Mobile Broadband Standard**. 2015. Accessed: Jul 20 2015. Available from Internet: <http://www.3gpp.org/article/lte>.

MCKEOWN, N. et al. OpenFlow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, ACM, v. 38, n. 2, p. 69–74, 2008.

MULLER, L. F. et al. Survivor: an Enhanced Controller Placement Strategy for Improving SDN Survivability. In: GLOBAL COMMUNICATIONS CONFERENCE (GLOBECOM), 2014 IEEE, 2014, Austin, TX, USA. **Proceedings...** Austin, TX, USA: IEEE, 2014. p. 1909–1915.

OPEN NETWORKING FOUNDATION. **OpenFlow Wireless**. 2011. Accessed: Mar 30 2015. Available from Internet: <http://archive.openflow.org/wk/index.php/OpenFlow_Wireless>.

OPEN NETWORKING FOUNDATION. **OpenFlow Switch Specification, Version 1.4.0**. 2013. Accessed: Apr 18 2015. Available from Internet: <http://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>.

OPEN VSWITCH. **Open vSwitch**. 2015. Accessed: Apr 18 2015. Available from Internet: <http://vswitch.org>.

OPENDAYLIGHT. **OpenDaylight: A Linux Foundation Collaborative Project**. 2015. Accessed: Mar 30 2015. Available from Internet: <http://www.opendaylight.org>.

PAASCH, C.; BARRE, S. et al. **Multipath TCP implementation in the Linux kernel**. 2015. Accessed: Apr 18 2015. Available from Internet: <http://www.multipath-tcp.org>.

PANDA, A. et al. CAP for Networks. In: SECOND ACM SIGCOMM WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, 2013, Hong Kong, China. **Proceedings...** New York, NY, USA: ACM, 2013. (HotSDN '13), p. 91–96.

PETRY, T. B.; SILVA, R. F. L.; BARCELLOS, M. P. Off the Wire Control: Improving the Control Plane Resilience through Cellular Networks. In: IEEE ICC 2015 - NEXT GENERATION NETWORKING SYMPOSIUM (ICC'15 (07) NGN), 2015, London, United Kingdom. **Proceedings...** London, United Kingdom: IEEE, 2015.

POX. **POX: A Python-based OpenFlow Controller**. 2015. Accessed: Mar 30 2015. Available from Internet: <http://www.noxrepo.org/pox/about-pox/>.

SALLAHI, A.; ST-HILAIRE, M. Optimal Model for the Controller Placement Problem in Software Defined Networks. **Communications Letters, IEEE**, v. 19, n. 1, p. 30–33, Jan 2015. ISSN 1089-7798.

SCHMID, S.; SUOMELA, J. Exploiting Locality in Distributed SDN Control. In: SECOND ACM SIGCOMM WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, 2013, Hong Kong, China. **Proceedings...** New York, NY, USA: ACM, 2013. (HotSDN '13), p. 121–126.

SCHULZ-ZANDER, J.; SARRAR, N.; SCHMID, S. AeroFlux: A Near-Sighted Controller Architecture for Software-Defined Wireless Networks. In: PRESENTED AS PART OF THE OPEN NETWORKING SUMMIT 2014 (ONS 2014), 2014, Santa Clara, CA. **ONS 2014**. Santa Clara, CA: USENIX, 2014.

TOOTOONCHIAN, A.; GANJALI, Y. HyperFlow: A distributed control plane for OpenFlow. In: 2010 INTERNET NETWORK MANAGEMENT CONFERENCE ON RESEARCH ON ENTERPRISE NETWORKING, 2010, San Jose, CA. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2010. (INM/WREN'10), p. 3–3.

TOOTOONCHIAN, A. et al. On Controller Performance in Software-Defined Networks. In: PRESENTED AS PART OF THE 2ND USENIX WORKSHOP ON HOT TOPICS IN MANAGEMENT OF INTERNET, CLOUD, AND ENTERPRISE NETWORKS AND SERVICES, 2012, San Jose, CA. **Hot-ICE'12**. Berkeley, CA: USENIX, 2012.

WANG, R.; BUTNARIU, D.; REXFORD, J. OpenFlow-based Server Load Balancing Gone Wild. In: 11TH USENIX CONFERENCE ON HOT TOPICS IN MANAGEMENT OF INTERNET, CLOUD, AND ENTERPRISE NETWORKS AND SERVICES, 2011, Boston, MA. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2011. (Hot-ICE'11), p. 12–12.

YAP, K.-K. et al. Blueprint for Introducing Innovation into Wireless Mobile Networks. In: SECOND ACM SIGCOMM WORKSHOP ON VIRTUALIZED INFRASTRUCTURE SYSTEMS AND ARCHITECTURES, 2010, New Delhi, India. **Proceedings...** New York, NY, USA: ACM, 2010. (VISA '10), p. 25–32.

YU, C. et al. FlowSense: Monitoring Network Utilization with Zero Measurement Cost. In: 14TH INTERNATIONAL CONFERENCE ON PASSIVE AND ACTIVE MEASUREMENT, 2013, Hong Kong, China. **Proceedings...** Berlin, Heidelberg: Springer-Verlag, 2013. (PAM'13), p. 31–41.

YU, M. et al. Scalable Flow-based Networking with DIFANE. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 40, n. 4, p. 351–362, aug. 2010. ISSN 0146-4833.

ZHANG, Y.; BEHESHTI, N.; TATIPAMULA, M. On Resilience of Split-Architecture Networks. In: GLOBAL TELECOMMUNICATIONS CONFERENCE (GLOBECOM 2011), 2011 IEEE, 2011. **Proceedings...** Houston, TX, USA: IEEE, 2011. p. 1–6.

ZHOU, X. et al. Mirror Mirror on the Ceiling: Flexible Wireless Links for Data Centers. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 42, n. 4, p. 443–454, aug. 2012. ISSN 0146-4833.

APPENDIX A — RESUMO ESTENDIDO DA DISSERTAÇÃO

A.1 Introdução

O paradigma de Redes Definidas por Software vem sendo um tema de pesquisa intenso da comunidade científica nos últimos anos. A separação dos planos de controle e de dados promete acelerar o ritmo de desenvolvimento, o controle logicamente centralizado tende a facilitar a operação da rede, e a consolidação de OpenFlow (MCKEOWN et al., 2008) como padrão de fato para o protocolo do plano de controle cria condições para interoperabilidade de dispositivos de vários fabricantes.

A.2 Contexto e Motivação

A separação dos planos, entretanto, criou uma forte relação de dependência entre o controlador e os dispositivos de encaminhamento, de maneira que partições na rede podem causar graves problemas de funcionamento (AKELLA; KRISHNAMURTHY, 2014). Falhas de enlace podem acabar isolando os dispositivos e o controlador, impedindo a difusão na alteração de políticas, rompendo a visão global da rede e, na falta de mecanismos de recuperação adequados, até tornar os dispositivos de encaminhamento temporariamente inoperantes.

A implantação do controlador como um sistema distribuído é uma estratégia frequente para prevenção da partição do plano de controle. O posicionamento de instâncias de controle em diversos pontos da topologia diminui a chance de isolamento entre dispositivos e controlador. Entretanto, essa estratégia traz desafios próprios de sistemas distribuídos no que diz respeito à sincronização das instâncias, de maneira a manter uma visão atualizada e consistente do estado da rede. Uma estratégia alternativa consiste em delegar aos dispositivos de encaminhamento tarefas que estão associadas ao plano de controle, buscando atenuar as consequências do isolamento ao invés de prevenir a partição. Porém, isso implica abdicar em parte da programabilidade centralizada da rede que é própria do paradigma de Redes Definidas por Software.

A.3 Arquitetura Proposta

Motivado pela importância da questão da partição do plano de controle, este trabalho avalia a viabilidade de uma arquitetura de plano de controle com enlaces adicionais em rede celular de dados, para maior resiliência à partições. Embora haja na literatura trabalhos que envolvem tanto Redes Definidas por Software como Redes sem fio, este é o primeiro trabalho que investiga o uso da rede sem fio como recurso de contingência para prevenir a partição do plano de controle. Para tanto, é possível imaginar estes enlaces mantendo a conectividade entre instâncias de controle (*east/westbound*), no caso de controladores distribuídos, ou entre dispositivos de encaminhamento e o controlador (*southbound*). O foco da dissertação está nessa última opção. Devido às características do ambiente de teste, a interface sem fio é associada ao dispositivo de encaminhamento. Uma vez que os serviços de rede celular costumeiramente mantém o dispositivo servido atrás de um NAT (Network Address Translation), é difícil iniciar uma conexão a tal dispositivo partindo de fora da subrede. Como em OpenFlow a conexão é tipicamente iniciada pelo dispositivo de encaminhamento, este é o ponto conveniente para posicionar a interface celular.

A avaliação toma duas premissas: (i) há boa qualidade de sinal para a interface celular 4G e (ii) o controlador é alcançável por tal interface. A primeira premissa é razoável em regiões metropolitanas. A segunda premissa levanta questões quanto a trafegar mensagens de controle pela rede pública, mas é possível aplicar medidas conhecidas de segurança como criptografia e SSL (Secure Sockets Layer) no canal de controle, conforme recomendado pela especificação do protocolo OpenFlow (OPEN NETWORKING FOUNDATION, 2013). A avaliação é focada em oportunidades e limitações desta arquitetura, baseada em três aspectos: (a) os efeitos das características de enlaces celulares sobre o tráfego OpenFlow, (b) o mecanismo de recuperação de falhas e migração de um enlace primário cabeado para um enlace sem fio, e (c) a operação de uma função de rede que depende de visão global da rede e que tenha um nível de interação limitado e configurável entre o dispositivo de encaminhamento e o controlador.

A.4 Avaliação

Para determinar os efeitos das características do enlace sem fio em tráfego OpenFlow, o primeiro conjunto de experimentos é realizado com trocas de mensagens do protocolo, variando a quantidade de conexões simultâneas e o tamanho das mensagens em ambas as direções. A métrica de tempo de ida e volta de cada troca de mensagens é baseada na saída da ferramenta *cbench* (TOOTOONCHIAN et al., 2012), desenvolvida com vista a avaliar o desempenho de controladores.

Este conjunto de experimentos corrobora a ideia de que o limitante inferior de latência em enlaces sem fio é significativo. Além disso, tanto o tamanho dos pacotes como o número de trocas de mensagem concorrentes pode aumentar a latência ao passo que a ocupação de banda é alcançada, ponto a partir do qual a diferença de banda entre o upload e o download afeta proporcionalmente o tempo de ida e volta das trocas de mensagens.

O procedimento de migração do plano de controle é experimentado com o desligamento da interface primária cabeada, em três condições: (a) com o dispositivo de encaminhamento em software Open vSwitch (OPEN VSWITCH, 2015) sem modificações, (b) com o dispositivo ligeiramente modificado para permitir a configuração de um intervalo menor para verificação de inatividade do canal de controle e (c) com o mesmo dispositivo e apoio de uma conexão de controle com Multipath TCP (PAASCH; BARRE et al., 2015). Nos dois primeiros casos, a migração depende de reconexão pela interface sem fio, enquanto o uso de múltiplos caminhos sob a abstração de MultipathTCP permite o preestabelecimento de subfluxos por ambas as interfaces. A métrica de tempo de migração considera a diferença entre o último pacote recebido pelo controlador antes de desativar a interface primária por cabo e o primeiro pacote recebido pela interface de contingência sem fio.

Este experimento deixa claro que o tempo de migração é dominado pelo intervalo de verificação de inatividade do plano de controle nas conexões de caminho único. Somado a ele, há um tempo adicional de restabelecimento da conexão OpenFlow pela interface de contingência. Com Multipath TCP, é possível realizar uma recuperação mais rápida, livre do tempo de reconexão. Entretanto, o seu uso depende de módulos de *kernel* que suportem a tecnologia nas duas pontas da conexão, e está sujeito ao comportamento dos dispositivos de rede ao longo do caminho em relação aos cabeçalhos de opção TCP, usados para controle de conexão Multipath TCP.

Para avaliar o funcionamento de uma aplicação de rede que dependa da visão global, o terceiro conjunto de experimentos é realizado sobre uma função de balanceamento de carga inspirada em (WANG; BUTNARIU; REXFORD, 2011). A capacidade de reação da aplicação é configurada com diferentes taxas de coleta de estatísticas e granularidades de regras de fluxo. A avaliação se baseia na eficiência do balanceamento, medida na ocupação das portas ligadas aos servidores para os quais a carga balanceada é direcionada, e na rapidez de entrega dos pacotes no plano de dados, medida pela vazão neste plano.

Com esta aplicação operando sobre o enlace sem fio, a carga é balanceada sem grande impacto na entrega dos pacotes de dados, e o balanceamento é, em média, eficiente. Entretanto, este comportamento favorável se deve em parte à política de balanceamento escolhida, que procura limitar o impacto da latência sobre o tempo de entrega de pacotes e desviá-lo para uma possível perda de eficiência no balanceamento. Para aplicações que dependem fortemente da instalação reativa de regras, o impacto de desempenho é mais significativo devido ao limiar inferior de latência dos enlaces sem fio.

A.5 Contribuições

A principal contribuição deste trabalho é fornecer uma análise da operação de tráfego OpenFlow em enlaces de rede celular que é pioneira, de acordo com o conhecimento do autor. Dentro dessa análise, merece destaque a indicação de que *os enlaces de contingência sem fio permitem manter o funcionamento de funções de rede sob certas circunstâncias*, a despeito da banda mais limitada e de latência superior aos enlaces cabeados. Entretanto, esta arquitetura atualmente não se credencia a substituir completamente uma infraestrutura cabeada de plano de controle. Sua aplicação é mais apropriada como uma arquitetura simples e de baixo custo para contingência do plano de controle, que visa manter uma visão global consistente do estado da rede e uma reatividade limitada para decisões de controle. Com a devida configuração e com o auxílio de tecnologias como Multipath TCP, os enlaces de contingência podem entrar em ação rapidamente na ocorrência de falhas nos enlaces primários.

A aplicabilidade da arquitetura resiliente proposta nesta dissertação é *fortemente relacionada aos requisitos das funções de rede e à maneira como as aplicações de controle são programadas*. Como regra geral, quanto mais uma dada função de rede bloqueia na ocorrência de eventos que dependem da intervenção do controlador, mais ela é afetada pela latência de um plano de controle em rede celular. Ademais, o impacto pode ser agravado em caso de tráfego intenso de controle, ocasionado, por exemplo, por um grande número de eventos concorrentes.

Em comparação com os mecanismos que visam tratar de partição no plano de controle, *esta arquitetura tem por objetivo manter uma visão consistente, centralizada e global da rede, ainda que com eventual atraso*. Controladores com instâncias distribuídas normalmente lidam com um compromisso entre oferecer o estado mais recente e manter tal visão global consistente, que também está sujeita a atraso devido à execução algoritmos de consenso. O favorecimento da visão mais recente é normalmente obtida ao se tirar proveito da localidade de algumas operações, que também está sujeita à maneira como as aplicações de controle são programadas. Os mecanismos de plano de dados são geralmente projetados para realizar tarefas sem depender de visão global, sendo assim complementares às demais propostas. Porém, isso implica tirar do controlador logicamente centralizado tal tarefa.

A.6 Trabalhos Futuros

Entre oportunidades de trabalhos futuros, pode-se destacar o potencial de aplicar o conceito de enlaces sem fio para comunicação entre módulos distribuídos de controladores logicamente centralizados ou para interconexão de diferentes controladores. A primeira opção está relacionada ao amadurecimento da pesquisa sobre características próprias de um plano de controle distribuído e à disponibilidade de controladores distribuídos de código livre, enquanto a segunda dependeria da convergência de algum padrão para comunicação entre controladores (*east/westbound*).

Outra linha possível é a exploração do posicionamento dos enlaces sem fio na topologia do plano de controle, considerando possibilidades como compartilhamento de enlace para canais de controle de diversos dispositivos, levando em conta o custo dos serviços e ocupação dos canais. O posicionamento do dispositivo sem fio em uma estação de controle é uma outra opção que pode estender este trabalho, condicionada à popularização no oferecimento de serviços de acesso a dispositivos celulares pela rede pública (isto é, com um endereço IP público e alcançável).

Dentre os experimentos realizados neste trabalho, a extensão natural da avaliação dos efeitos do enlace sem fio seria enfraquecer as premissas de qualidade de sinal, realizando uma avaliação a longo prazo para identificar padrões de variação do sinal e caracterizar seus efeitos. Outro adendo seria a troca do recurso de geração de mensagens por traços de tráfego OpenFlow obtido em ambiente de produção, quando disponíveis. Para os experimentos de migração, a extensão mais interessante seria observar e avaliar o impacto da migração em operações do plano de dados.

A avaliação do impacto em funções de rede foi baseada em uma aplicação de balanceamento de carga, que depende de visão do estado global e pode ser projetada com variado nível de dependência à reatividade do controlador. Ainda assim, a avaliação pode ser enriquecida com a exploração de outras funções que tiram proveito de visão global da rede, tais como sistemas de detecção de intrusão ou roteamento dinâmico baseado em políticas. O desafio começaria por identificar o grau de impacto de um possível atraso na visão do estado da rede sobre essas funções.

APPENDIX B — SOURCE CODE EXCERPTS

B.1 Source Code of Changes to cbench

This appendix presents the output of the *diff* command between each original repository file and our modified version. These changes enable a new command-line parameter specifying the size of the payload of an IP packet embedded inside each *packet in*. The original repository is hosted at: *git://gitosis.stanford.edu/oflops.git*

Diff to oflops/cbench/cbench.c

```
47a48
>     {"payload-size", 'z', "Payload␣size␣in␣the␣IP␣packet␣embedded␣in␣the␣packet-in",
    MYARGS_INTEGER, {.integer = 64}},
258a260
>     int     payload_size = myargs_get_default_integer(my_options, "payload-size");
328a331,333
>             case 'z':
>                 payload_size = atoi(optarg);
>                 break;
347a353
>                 "␣␣␣payload␣size␣is␣%d␣bytes\n"
361a368
>                 payload_size,
391c398
<         fakeswitch_init(&fakeswitches[i],dpid_offset+i,sock,BUFLEN, debug, delay, mode,
    total_mac_addresses, learn_dst_macs);
---
>         fakeswitch_init(&fakeswitches[i],dpid_offset+i,sock,BUFLEN, debug, delay, mode,
    total_mac_addresses, learn_dst_macs, payload_size);
```

Diff to oflops/cbench/fakeswitch.c

```
28c28
< static int make_packet_in(int switch_id, int xid, int buffer_id, char * buf, int buflen, int
     mac_address);
---
> static int make_packet_in(int switch_id, int xid, int buffer_id, char * buf, int buflen, int
     mac_address, int payload_size);
54c54
< void fakeswitch_init(struct fakeswitch *fs, int dpid, int sock, int bufsize, int debug, int
    delay, enum test_mode mode, int total_mac_addresses, int learn_dstmac)
---
> void fakeswitch_init(struct fakeswitch *fs, int dpid, int sock, int bufsize, int debug, int
    delay, enum test_mode mode, int total_mac_addresses, int learn_dstmac, int payload_size)
65c65,66
<     fs->probe_size = make_packet_in(fs->id, 0, 0, buf, BUFLEN, fs->current_mac_address++);
---
>     fs->payload_size = payload_size;
>     fs->probe_size = make_packet_in(fs->id, 0, 0, buf, BUFLEN, fs->current_mac_address++, fs
    ->payload_size);
293c294
< static int make_packet_in(int switch_id, int xid, int buffer_id, char * buf, int buflen, int
      mac_address)
---
> static int make_packet_in(int switch_id, int xid, int buffer_id, char * buf, int buflen, int
      mac_address, int payload_size)
296a298,301
>     payload_size = payload_size < 34 ? 34 : payload_size;
>     int ip_size = payload_size - 14;
>     int frame_size = ip_size + 14;
>     int packetin_size = frame_size + 18;
298,299c303,308
<                        0x97,0x0a,0x00,0x52,0x00,0x00,0x00,0x00,0x00,0x00,0x01,
<                 0x01,0x00,0x40,0x00,0x01,0x00,0x00,0x80,0x00,0x00,0x00,
---
>                        0x97,0x0a,
>                 (char)((packetin_size >> 8) & 0xff),(char)(packetin_size & 0xff),//packetin
     size was 0x00,0x52,
>                 0x00,0x00,0x00,0x00,0x00,0x00,0x01,
>                 0x01,
>                 (char)((frame_size >> 8) & 0xff),(char)(frame_size & 0xff),//frame length was
     0x00,0x40,
>                 0x00,0x01,0x00,0x00,0x80,0x00,0x00,0x00,
301,302c310,315
<                 0x00,0x00,0x32,0x00,0x00,0x00,0x00,0x40,0xff,0xf7,0x2c,
<                 0xc0,0xa8,0x00,0x28,0xc0,0xa8,0x01,0x28,0x7a,0x18,0x58,
---
>                 0x00,
>                 (char)((ip_size >> 8) & 0xff),(char)(ip_size & 0xff),//ip length was 0x00,0x32
     ,
>                 0x00,0x00,0x00,0x00,0x40,0xff,
>                 0x00,0x00,//checksum was 0xf7,0x2c,
>                 0xc0,0xa8,0x00,0x28,0xc0,0xa8,0x01,0x28};
> /*                     ,0x7a,0x18,0x58,
306c319,332
<     assert(buflen> sizeof(fake));
---
> */
>     char payload[payload_size - (14 + 20)];
>     int i;
>     for (i = 0; i < sizeof(payload); i++) {
>         payload[i] = 0x00;
>     }
>     assert(buflen> (sizeof(fake) + sizeof(payload)));
>     long sum = 0;
>     for (i = 32; i < (32 + 20); ) {
>         sum += (fake[i++] & 0xff) << 8;
>         sum += (fake[i++] & 0xff);
>     }
>     sum = (~((sum & 0xffff) + (sum >> 16))) & 0xffff;
>     const char checksum[2] = {(sum >> 8) & 0xff, sum & 0xff};
```

```
307a334,335
>       memcpy(buf + 42, checksum, 2);
>       memcpy(buf + sizeof(fake), payload, sizeof(payload));
319c347
<       return sizeof(fake);
---
>       return (sizeof(fake) + sizeof(payload));
497c525
<               count = make_packet_in(fs->id, fs->xid++, fs->current_buffer_id, buf, BUFLEN, fs
    ->current_mac_address);
---
>               count = make_packet_in(fs->id, fs->xid++, fs->current_buffer_id, buf, BUFLEN, fs
    ->current_mac_address, fs->payload_size);
```

Diff to oflops/cbench/fakeswitch.h

```
41a42
>       int payload_size;
55a57
>  * @param payload_size
57c59
< void fakeswitch_init(struct fakeswitch *fs, int dpid, int sock, int bufsize, int debug, int
    delay, enum test_mode mode, int total_mac_addresses, int learn_dstmac);
---
> void fakeswitch_init(struct fakeswitch *fs, int dpid, int sock, int bufsize, int debug, int
    delay, enum test_mode mode, int total_mac_addresses, int learn_dstmac, int payload_size);
```

B.2 Source Code of Changes to POX

This appendix presents the output of the *diff* command between the original repository file and our modified version. These changes are based on release *0.2.0*, branch *carp*. It simply truncates the *strip-VLAN* action to four bytes by removing the padding zeroes. The original repository is hosted at: *https://github.com/noxrepo/pox*

Diff to pox/pox/openflow/libopenflow_01.py

```
1643c1643
<       packed = struct.pack("!HHi", self.type, len(self), 0)
---
>       packed = struct.pack("!HH", self.type, len(self))
1649c1649
<       offset = _skip(raw, offset, 4)
---
>       offset = _skip(raw, offset, 0)
1655c1655
<       return 8
---
>       return 4
```

B.3 Source Code of Changes to Open vSwitch

This appendix presents the output of the *diff* command between each original repository file and our modified version. These changes are based on release *v1.9.0*. They enable configuring the probe interval down to 1 second, instead of the original lower bound of 5 seconds. The original repository is hosted at: *https://github.com/openvswitch/ovs*

Diff to ovs/lib/rconn.c

```
243c243
<     rc->probe_interval = probe_interval ? MAX(5, probe_interval) : 0;
---
>     rc->probe_interval = probe_interval ? MAX(1, probe_interval) : 0;
```

Diff to ovs/ofproto/connmgr.c

```
1138c1138
<     probe_interval = c->probe_interval ? MAX(c->probe_interval, 5) : 0;
---
>     probe_interval = c->probe_interval ? MAX(c->probe_interval, 1) : 0;
```

APPENDIX C — ACCEPTED PAPER AT IEEE ICC

This appendix contains the paper "Off the Wire Control: Improving the Control Plane Resilience through Cellular Networks". It was presented at the IEEE International Conference on Communications.

- Title: Off the Wire Control: Improving the Control Plane Resilience through Cellular Networks
- Conference: 2015 IEEE International Conference on Communications (ICC)
- URL: http://dx.doi.org/10.1109/ICC.2015.7249167
- Date: June 8 – 12, 2015
- Location: London, UK

Off the Wire Control: Improving the Control Plane Resilience through Cellular Networks

Tobias Petry, Rafael da Fonte Lopes da Silva,
Marinho P. Barcellos
Federal University of Rio Grande do Sul, Institute of Informatics, Porto Alegre, RS, Brazil
Email: {tbpetry, rflsilva, marinho}@inf.ufrgs.br

Abstract—**Software Defined Networks simplify network programmability by detaching the control plane from forwarding devices and deploying it into a logically centralized controller. While this allows a clearer separation of concerns, it also creates a dependency between them. Failures in the control plane break the controller view of the network state and could render the network unusable if forwarding devices cannot be reached. The relevance of this problem has led to a range of proposals, including physical distribution of controller instances and delegation of concerns to forwarding devices. In this paper, we propose and evaluate an architecture that leverages cellular data networks (4G) as control plane backup links. No previous work has explored this idea, despite the recent research intersecting SDN and wireless networks. Our experiments answer three research questions: (i) *How is the behavior of control plane traffic affected by the characteristics of cellular links*, (ii) *how quickly is the control plane handed over to the backup link when a failure occurs* and (iii) *how well do network functions that rely on a snapshot of the network state behave on such an architecture*. Our evaluation shows that, despite the expected higher latency of cellular links, this architecture maintains partial functionality of tasks that depend on global network awareness when failures occur in primary links in a simple, affordable fashion.**

I. Introduction

Over the past few years, the Software Defined Networking (SDN) paradigm has gained unprecedented strength within the scientific community. The split of the control and data planes is expected to increase development pace and to ease the job of network operation, through the separation of concerns and logically centralized control, respectively. As *OpenFlow* [1] consolidates as a *de facto* standard for the SDN control plane protocol, the interoperability between compliant forwarding devices – commonly referred to as OpenFlow *switches* – and controller software allows for a vendor-neutral system. Considering these benefits, research has been devoted to applying SDN into other areas, including wireless networks.

As a consequence of the separation between the planes, a dependency relation was created among entities which are now potentially physically separated – namely, the controller and the switches. Therefore, partitions in the network can severely disrupt functionality. For instance, link failures can isolate devices from their corresponding control element, preventing them from being aware of policy updates, breaking the global visibility of the network and, in the absence of failover mechanisms (e.g. fail standalone mode in hybrid switches), even render the device temporarily unusable.

Motivated by these issues, this work evaluates the suitability of a control plane architecture featuring additional cellular data links for improved resilience in face of network partitions in the control plane. We focus our evaluation on three aspects, namely: (a) the effects of the cellular link characteristics on OpenFlow traffic, (b) the failover mechanism from the primary wired links to the backup wireless link, and (c) the operation of a network function that uses global network state awareness and can have a limited, configurable level of interaction between the forwarding device and the controller – a feature that we refer to as *reactiveness*.

The main contribution of our work is to provide a novel analysis of the operation of OpenFlow control plane traffic over cellular links. We find evidence that, in spite of providing lower bandwidth and higher latency than wired links, cellular links can still allow an OpenFlow-based network to maintain the operation of network functions under certain circumstances without requiring modifications to the underlying devices.

The remainder of the paper is organized as follows: in the next section we discuss related work (II) in SDN with wireless networks and control plane connectivity. Section III describes our proposed architecture. The methodology is briefly presented in Section IV. Then, we analyze our approach according to the three questions introduced in the abstract in Sections V, VI and VII. Finally, Section VIII contains our final remarks and planned future work.

II. Related Work

In this paper, we evaluate an architectural setup that employs cellular network links in the maintenance of control plane connectivity in SDN. We organize related work in two broad topics. First, those that concern wireless networks in the context of SDN. Second, those that tackle the loss of connectivity of the control plane by attempting either to prevent such a loss or to attenuate its consequences.

A. Wireless networks and SDN. There have been different approaches combining wireless infrastructure and SDN. OpenRoads [2] proposes decoupling the infrastructure from the network services into wireless mobile networks. In [3], the idea is to simplify the design and management of cellular networks using SDN. The work of [4] employs wireless links and reflective surfaces on the facility roof to increase the trajectory space and allow for more simultaneous connections between pairs of nodes.

In [5], an extension for adding Media Independent Management into the OpenFlow protocol is proposed, so that link condition information (e.g. signal strength, power save mode) can be leveraged by the controlling mechanisms. It is worth

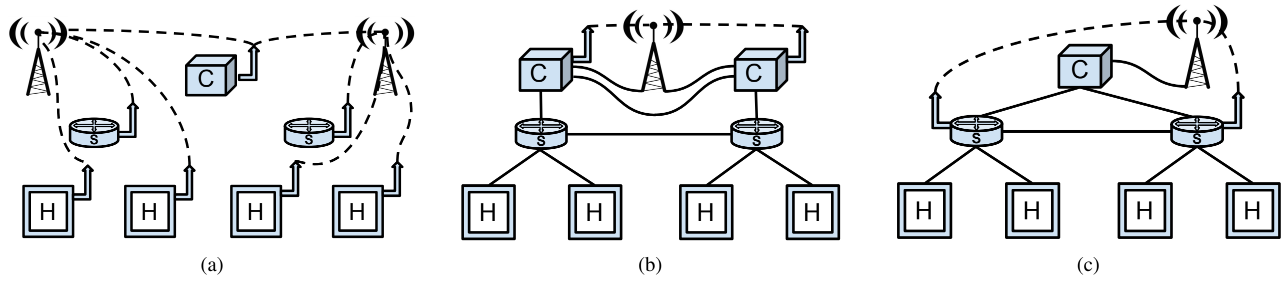


Fig. 1. Cellular links in SDN: (1a) shared data plane traffic and physically in-band control plane traffic, (1b) east/westbound control plane traffic among controller instances, (1c) southbound control plane traffic between forwarding device and controller. *C* stands for controller, *S* for switch and *H* for host.

noticing that, while the work at [5] aims at being independent to the link network technology, the experimental setup was deployed with wireless links solely in the data plane – the control channels were wired Ethernet. Aeroflux [6] presents a hierarchical network design for a software-defined WiFi network, featuring a global controller and subordinate near-sighted controllers. The evaluation of a prototype indicates reduction in the control traffic.

B. Control plane connectivity. The loss of control plane connectivity could potentially compromise network functionality. For example, a forwarding device incapable of communicating with its controller is neither able to receive instructions for new flows nor become aware of updates in access control policies. Current methods proposed in the literature to address these issues can be categorized into two architectural strategies: 1) physical distribution of the controller and 2) delegation of concerns to forwarding devices.

B1. Physical distribution of the controller. Although the proposals of physically distributed controllers often highlight scalability as its main contribution [7], partition tolerance is still an important consideration [8], [9]. In fact, the positioning of the distributed instances is a topic of research on its own [10]–[13]. Keeping a logically centralized behavior means synchronizing the state among control instances, and the trade-offs of the CAP theorem (consistency, availability and partition tolerance) apply to SDN as well [14]. The synchronization mechanisms are particular to each proposal, ranging from replicated relational databases to key-value stores. In addition, despite the increasing attention and interest seen in this area, the offer of open-source distributed controllers is still limited. Worthy of note, however, are OpenDaylight [15] and ONOS [16], both featuring shared state among cluster nodes.

B2. Delegation of concerns to forwarding devices. The switches built for Google's B4 WAN [17] have an OpenFlow software layer atop hardware that is able to directly run routing protocols, in the trend of OpenFlow's optional *fail-standalone* mechanism [18] (since version 1.1). DIFANE [19] proposes turning some devices into *authority switches* that can react to incoming flows in unmodified devices. In [20], ideal forwarding connectivity can be achieved at the data plane by using small and fast operations that store state in the packet headers, based in the properties of Directed Acyclic Graphs. In contrast to these solutions, in the proposed approach no modifications to the forwarding devices, to the controller or to the OpenFlow protocol are required for keeping control plane connectivity in the event of failures in primary wired links.

III. Off the Wire Control Plane

Considering the aforementioned aspects of dependency among controllers and forwarding devices, we propose a novel architecture for the SDN control plane consisting of additional cellular links (e.g. 4G, LTE) between forwarding devices and controller. This approach improves avoidance of control plane network partitions in a transparent fashion to the network elements, using low cost, off-the-shelf hardware. By keeping control plane connectivity under failure of the primary link, the architecture allows for maintaining network management activities and partial functionality.

There are three settings in which additional cellular links could be used to extend the network: a) for shared data plane traffic and physically in-band control plane traffic, b) for east/westbound control plane traffic among controller instances, and c) for southbound control plane traffic between forwarding device and controller. These alternatives are depicted in Figure 1.

The first alternative, shown in Figure 1a, has already been explored in works such as [3] and [5], though the approach assumed network infrastructures built exclusively with cellular links. A similar approach is to leverage the SDN benefits to control and manage very dense heterogeneous wireless networks [21] – for instance, by employing a multi-tier controller hierarchy for a more efficient management of medium usage.

The second alternative (Figure 1b), albeit promising, still depends on the convergence of an east/westbound standard, as well as on the maturing of open distributed controllers. As discussed in Section II, previous work on controllers with distributed instances either consists of closed-source solutions (e.g. [7]) or relies on non-standard synchronization frameworks such as distributed file systems or key-value pair stores. Works such as [8] have already noted certain restrictions in terms of control application behavior and policy decisions towards state synchronization, notably for scalability reasons related to event handling and device probing. Although we do not explore this possibility in the present paper, the use of backup cellular links for keeping state synchronized among separate controller instances has good potential.

The third alternative, shown in Figure 1c, is to extend the control plane connectivity through additional wireless links between the forwarding devices and the controller. We focus our study on this approach, and describe the methodology to perform the evaluation in the next section.

IV. Methodology

In our study, we aim to answer three research questions: how the behavior of the control plane is affected by characteristics of cellular links; how quickly the control plane is handed over to a backup link when a failure occurs; and how well do network functions that rely on a snapshot of the network state behave on such an architecture. We consider the following two assumptions for the evaluation: (i) there is sufficient signal availability on the site of the hosts (i.e. forwarding devices) connected to 4G interfaces, and (ii) a host can reach the controller listening interface through the 4G links. It is reasonable to consider the first assumption given that signal coverage on metropolitan areas is widely available. While the second assumption might raise concerns about using public networks for the control traffic, it is feasible to apply known security measures such as cryptography and SSL, as recommended by the OpenFlow protocol specification [18].

In each of the experimental settings we describe, Host 1 is a 2 GHz Core 2 Duo machine with 4GB RAM running VMware Player 6 on Windows 7 32 bits. The virtual machines receive a single processor core and have 1GB RAM available. Cbench [22] is installed on Ubuntu 14.04. We used Mininet VM [23] image version 2.1.0 (with Ubuntu 13.04), with either the bundled Open vSwitch 1.9.0 or its recompiled version with altered lower bound for the inactivity probe. Host 2 runs the POX [24] controller 0.2.0 (carp) on a 1.8 GHz (x 4) Core i3 with 4GB RAM, on Ubuntu 14.04 64 bits.

V. Effects of cellular link in OpenFlow traffic

A. *How is the behavior of control plane traffic affected by the characteristics of cellular links?*

Some features of cellular links – namely, latency and bandwidth – are expected to be limiting factors to be dealt with in actual control plane network deployments. We devised a set of experiments that stress the communication between controller and forwarding devices, in an attempt to measure how strong this limitation is in comparison to wired connections.

With OpenFlow, the control plane communication is achieved through a TCP connection between each forwarding device and the controller. To understand the effects of link latency and bandwidth, we measure the response times for OpenFlow exchanges in two scenarios, each with the conversation initiated at either one of the sides. In the first scenario, we use the sequence of messages required to install a rule which will match a new flow arriving at a device, as follows. A *packet-in* message describing the new flow – typically, containing the header of the first packet in the flow – is sent to the controller. The controller is then responsible for processing the message, and sending a reply with instructions of what the device should do with the packet. For instance, this reply could be a *packet-out* – instructing where to forward the packet. For this experiment we use *cbench*, a tool for evaluation of controller performance through generation of OpenFlow traffic. Figure 2 illustrates the setup for this experiment.

In order to avoid adding processing overhead to our experiment, we run the controller using a synthetic application with a simple behavior: once the controller receives a *packet-in*, the control application immediately sends a reply with a *packet-out* message with no actions, regardless of the *packet-in* content – essentially an instruction to drop that packet. The *cbench* tool runs in latency mode and emulates the data plane. It performs *N* concurrent message exchanges. In each of them, a *packet-in* message is sent to the controller, and the corresponding *packet-out* message is then waited for. Thus, *N* represents the number of concurrent message exchanges (one per forwarding device emulated).

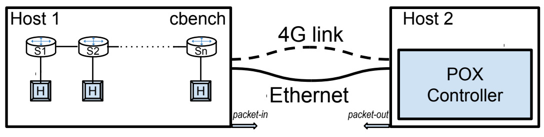


Fig. 2. Two hosts are used in this and the following experiments. They are connected either by Ethernet or a 4G link (one at a time). A set of *n* switches generating messages is emulated by running *cbench* at Host 1, and the POX controller responds from Host 2.

We linearly increase the number of simultaneous *packet-in* – *packet-out* exchanges, all sharing the same connection with the controller. We measure the mean time for completing each exchange over 50 seconds, both using 4G and Ethernet for the control plane link. While it is expected that the communication over the wireless link will have a higher latency, we intend to identify whether there is a stable, bounded time for this exchange and what are the effects of having several devices simultaneously performing such interactions.

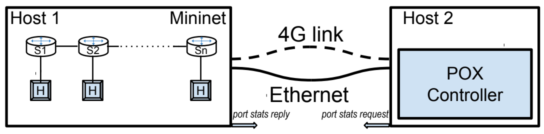


Fig. 3. In this experiment, Host 2 runs POX, while Mininet is used at Host 1 to emulate a set of switches receiving requests originated at the controller.

We evaluate the latency of exchanges originated at the controller through the mechanism of statistics collection provided by OpenFlow. In the experiment, the controller sends a *port-stats-request* message to a forwarding device and waits for a *port-stats-reply*, as illustrated in Figure 3. Similarly to the previous experiment, we vary the number of forwarding devices – they are connected in series, under Mininet – and measure the mean round trip time using the two types of channel, over 50 seconds.

B. *Latency lower bound is significant, packet size matters*

Results in Figures 4 and 5 indicate that the exchange time for *packet-in* conversations grows steadily with the number of emulated switches, in a similar rate for both the wired and wireless link experiments. The exchange time is clearly lower bounded by the channel latency, as the difference between the two media for each set of parameters varies between 115 and 130 ms. This is not true for the second experiment, as the mean exchange time grows much faster with a 4G control plane. While there is a difference in the experiment setup – with a network being emulated instead of only switches – this is not the bottleneck, as we observe a much lower difference between the Ethernet measurements of both experiments.

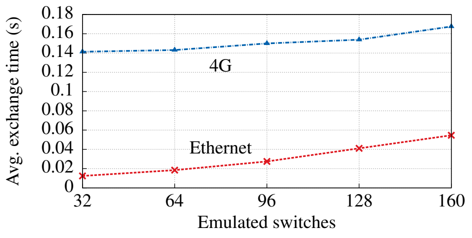


Fig. 4. OpenFlow message exchanges started at the switch.

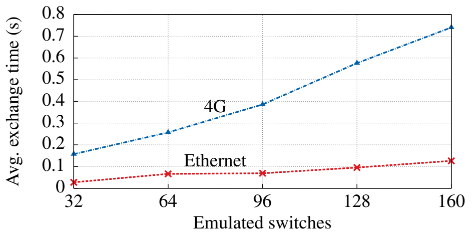


Fig. 5. OpenFlow message exchanges started at the controller.

As expected, the packet size plays a more prominent role in the measured results. By observing the packets on the control interface, we find that both the *packet-in* and the *packet-out* messages exchanged in the first experiment have smaller payload size than their counterparts in the same direction in the second experiment – respectively, *port stats reply* and *port stats request*. In the case of messages sent by the switches, the payload of each *packet-in* is 82 bytes long, against either 324 or 428 bytes of the *port stats reply* payload (switches in both ends of the serial topology have one less port, and therefore send messages with smaller payloads). As for messages originated at the controller, the payload difference is small – 16 bytes for the *packet-out* and 20 bytes for *port stats requests*. Packet size variation is the most likely cause for the higher latency impact in the port statistics exchange experiment. In addition, frequently the *packet-in* payloads are grouped into a single TCP message, reducing the overhead of lower layer header bytes. Conversely, no grouping occurs in the *port stats replies*.

VI. Failover to the Backup Cellular Link

A. How quickly is the control plane handed over to the backup link when a failure occurs?

Recall that the scenario we are investigating is aimed towards ensuring connectivity between forwarding devices and controller instances in the face of network partitions. For a wide variety of applications that rely on SDN networks, it is of great importance to ensure good responsiveness of the underlying network devices, and guaranteeing fast failover times on these devices is crucial for minimizing the impact on data plane operation (e.g. packet losses, unacceptable latencies).

Considering that the cellular link is to be employed as a backup for a wired control plane channel, we need to evaluate how quickly the control plane is healed through the wireless link when the primary link ceases to operate. We measure the total time, or how long it takes for the forwarding device to start operating over the backup link.

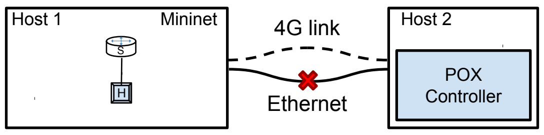


Fig. 6. In this experiment, emulated host and switch run in Mininet at Host 1, and the POX controller at Host 2. As before, there are two links between the hosts, and the Ethernet one is deactivated during communication to measure how long it takes for the control plane to recover.

As shown in Figure 6, we deploy a single host connected to a switch on Mininet. The emulated host sends a packet to the emulated switch every 100 ms to trigger *packet-in* events. The control application always responds with an actionless *flow-mod*, so that *packet-in* messages keep being generated.

Failover is attained by configuring gateways for both media. We use a lower metric on the gateway corresponding to the wired interface, to ensure that traffic will be primarily directed through it. We insert a failure by taking down the Ethernet network interface. In our experiments, we insert the failure in the first hop for the sake of simplicity. Failures along the path could be detected on the TCP connection with an auxiliary mechanism, out of the scope of this work – for instance, an user space script could be configured to probe the controller with *ping* and alter the gateway setup with arbitrary periodicity and probe interval, establishing the control plane communication over the disjoint backup path. Over 30 runs, we measure at the controller the time between the arrivals of the last *packet-in* from the primary connection and the first *packet-in* from the new one, established through the 4G link.

Open vSwitch allows the configuration of an inactivity probe interval for the control plane channel. When the communication is idle for the configured amount of time, the switch probes the controller. If no answer is received after this same amount of time, it tries to reestablish the connection. Open vSwitch has both a default and minimum value of 5 seconds probe interval – this is true for the default Open vSwitch version 1.9.0, bundled in the Mininet VM 2.1.0, as well as up to its most recent build (by the time of writing), 2.3.0. We compare the failover times between an unmodified Open vSwitch and a recompiled version with the probe interval lower bound reduced to 1 second.

B. Failover time is dominated by probe interval

The failover time is lower bounded to 10 seconds when using an unmodified version of the software switch. The timeline in Figure 7 illustrates the failover process. By altering two lines of code in the OpenFlow connection manager of the switch, we brought this hard-coded limit down to 1 second. We are aware that reducing the inactivity probe interval increases the probability of false positives in detecting loss of the control plane connection. Nevertheless, this is a trade-off that can be determined by the network administration – reducing the lower bound does not prevent configuring the actual inactivity probe to a higher value. We measured the perceived down time from the controller point of view. Table I shows the minimum, mean, maximum time and standard deviation between the last received communication from the OpenFlow switch and the first *packet-in* message after recovery in (a) an unmodified

TABLE I. Failover process results (s)

	Minimum	Average	Maximum	Std. Dev.
Open vSwitch 5 sec. probe	11.491	12.487	14.864	0.865
Open vSwitch 1 sec. probe	2.600	3.560	8.673	1.434

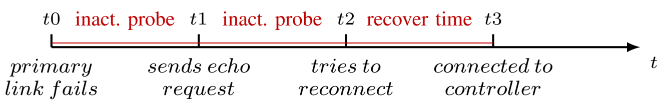


Fig. 7. The failover measurements in Table I refer to the highlighted interval between *t0* – *t3*, including twice the configured inactivity probe. *t0* – *t1* is the idle control plane time, after which the device probes the controller with an *echo request*. It waits for the same amount of time for an *echo reply* (*t1* – *t2*). Finally, on *t2* the switch tries to reach the controller through the backup link and the first *packet-in* message arrives on *t3*.

Open vSwitch 1.9.0 with a 5 second inactivity probe interval and (b) in a 1 second probe deployment.

The experiment result shows that, once a failure in the primary connection is detected (twice the probe interval), the time for reestablishing the connection with the controller stays within 10 seconds. Each part of the message exchange specified by the OpenFlow protocol upon connection establishment (e.g. features request and reply) can be affected by varying latency of the backup link, resulting in the showcased range of recovery times. It must be noted that this architecture is expected to provide a backup link for keeping global visibility, and not to be suitable for extremely reactive network functions.

VII. Network Function & Off the Wire Control

A. *How well do network functions that rely on a snapshot of the network state behave on such an architecture?*

One of the most attractive features of OpenFlow is the ability to perform decisions based on a global view of the network, simplifying network programmability and consequently the deployment of network policies. The proposed architecture, enhanced with wireless links, can also be seen as an effort towards the maintenance of such network visibility within controller instances. The third and final question attempts to verify whether network applications which depend upon global visibility can still perform well under the conditions imposed by this alternative medium.

We use a load balancing application as a means to answer the third question. Load balancing is a network function usually employed as an example of application that can be based on global network state awareness [6], [25], [26]. We follow an address prefix scheme, similar to that of [25], where a binary tree is built for partitioning the space of source IP addresses among different server instances. Wildcarded rules are then used to configure the virtualized Open vSwitch.

Figure 8 shows three emulated hosts connected to the emulated switch. One host produces traffic directed to a given destination IP address, which must be balanced among the server hosts. A total of 6000 UDP packets are sent, scheduled according to a Poisson process with a mean interval of 10 ms. The size of each packet is exponentially distributed with a mean of 1024 bytes, upper-bounded by the Ethernet frame payload limit (1500 bytes including IP and UDP headers). The source IP is spoofed from a range of 1024 sequential addresses, in order to allow the range-based load balancing to take place. For all conducted experiments, the generated load is the same. The traffic is directed towards two server hosts, according to the load balancing policy enforced by the control application on the forwarding device.

The set of parameters on the control application configures the load balancer *reactiveness* features. The controller periodically requests statistics to the OpenFlow switch, becoming aware of the current state of the servers corresponding ports. The interval between each statistics request message is varied between 0.5s, 1s and 2s. A longer interval decreases control traffic at the cost of possibly reacting too late, due to stale state. Reactiveness is particularly configured by adjusting the granularity of the rules. The full range of distinct source IP addresses of the experiment (1024) is subdivided in 32, 64 and 128 sub-ranges. For instance, dividing the range in 128 sub-ranges causes each installed rule to match 8 addresses.

Intuitively, a more *reactive* controller should perform load balancing better than a more *proactive* one, at the cost of higher control link occupancy. In this experiment, we attempt to assess the degree of reactiveness the control instance can reach before data plane services become visibly degraded under the effects of a cellular control link. We measure the negative impact to load balancing by comparing the relative usage of the server ports. Additionally, we obtain the mean data plane throughput, to assess whether the control reaction time for forwarding decisions causes a significant delay in packet delivery.

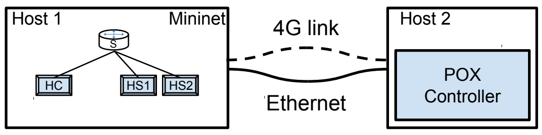


Fig. 8. *HC* produces traffic with different IP sources towards either emulated host server (*HS1* or *HS2*), passing through the emulated switch. Routing rules are configured according to the load balancer running at POX in Host 2. *HC*/*HS* stand for client/server.

B. *Load is balanced seamlessly and, on average, efficiently*

The results for relative usage of server ports are presented as RMSE (Root-Mean-Square Error) box plots (Figure 9). For instance, a value of 0% means that the load is equally balanced among the two servers. Meanwhile, 100% stands for a case where the load is completely forwarded to one of the servers. The usage is measured in intervals of 0.5s.

The higher latency in the 4G control plane links has caused occasional late reactions in the load balancing process, as illustrated by the outliers in Figure 9b. The experiments conducted with an Ethernet control link feature a more consistent behavior (Figure 9a). Nevertheless, the average load balancing effectiveness is very similar in both cases. Since the variation in the number of flow sub-ranges and in the periodicity of statistics collection show little significance in the measured results, we observed that the major drawback of using 4G links in the control plane for a load balancing function is the occasional mistaken forwarding decision due to a delayed view of the data plane state.

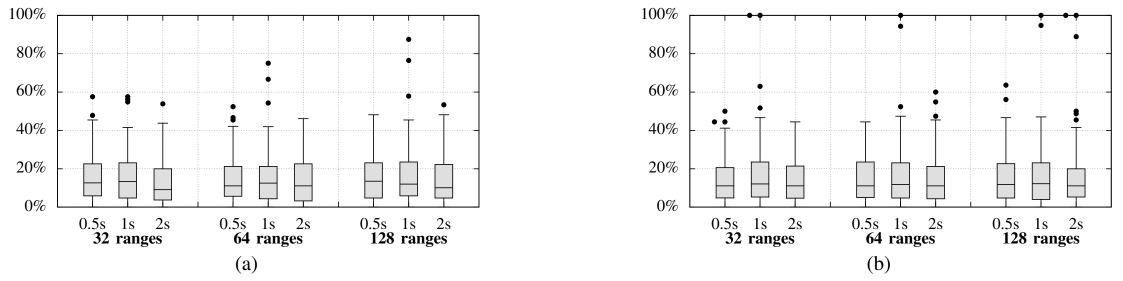


Fig. 9. RMSE of the relative usage of two server ports with a control plane link over (9a) Ethernet and (9b) 4G. In each plot, the number of IP address subranges and the periodicity of port status querying by the controller are varied to change the *reactiveness* of the load balancing function.

TABLE II. EFFECT OF CONTROL LATENCY ON DATA THROUGHPUT

Flow ranges		32	64	128	256	512
Ethernet	kb/s	587.01	593.98	584.46	568.98	590.72
	Rules	156	880	2211	3700	4650
4G	kb/s	593.44	574.25	596.31	588.68	588.55
	Rules	224	1078	2328	3719	4624

Analyzing the throughput measurements, we see that the load balancing application has little or no impact in the timeliness of data plane packet delivery. Table II shows that, even with much higher rate of flow installations due to smaller address sub-ranges, the resulting throughput is consistent among all measurements in both media.

VIII. CONCLUSION

To the best of our knowledge, this work features the first analysis of the operation of OpenFlow control traffic over cellular links. It is challenging to answer the discussed questions generally, and our experimental study provides the first insights towards the answers. Our evaluation shows that, in spite of providing lower bandwidth and higher latency than wired links, cellular links can allow an OpenFlow-based network to maintain operation of network functions under certain circumstances. In particular, a wireless backup control architecture is a simple, inexpensive way of maintaining global network state awareness and limited reactiveness requirements for control plane decisions.

As for future work, we see great potential in applying the concept of wireless links for communication between physically distributed modules of logically centralized controllers or for interconnecting different controller instances. While the former alternative should emerge as soon as new open-source distributed controllers become available, the latter would depend on the convergence of an eastbound API standard.

ACKNOWLEDGMENT

This work has been supported by MCTI/CNPq/Universal 14/2014 (Phoenix, 460322/2014-1) and MCTI/CNPq/CT-ENERG 33/2013 (ProSeG, 404958/2013-3).

REFERENCES

[1] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Comm. Rev.*, vol. 38, no. 2, 2008.

[2] K.-K. Yap et al., "Blueprint for introducing innovation into wireless mobile networks," in *ACM SIGCOMM VISA workshop*, 2010.

[3] L. E. Li, Z. M. Mao, and J. Rexford, "Toward software-defined cellular networks," in *IEEE EWSDN*, 2012.

[4] X. Zhou et al., "Mirror mirror on the ceiling: flexible wireless links for data centers," in *ACM SIGCOMM*, 2012.

[5] C. Guimarães, D. Corujo, and R. L. Aguiar, "Enhancing OpenFlow with Media Independent Management capabilities," in *IEEE ICC*, 2014.

[6] J. Schulz-Zander, N. Sarrar, and S. Schmid, "AeroFlux: A near-sighted controller architecture for software-defined wireless networks," *Open Networking Summit*, 2014.

[7] T. Koponen et al., "Onix: a distributed control platform for large-scale production networks," in *USENIX OSDI*, 2010.

[8] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *USENIX INM/WREN*, 2010.

[9] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *ACM HotSDN*, 2012.

[10] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *GLOBECOM*, 2011.

[11] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, Sep. 2012.

[12] M. F. Bari et al., "Dynamic controller provisioning in software defined networks," in *CNSM*, Oct. 2013.

[13] L. Muller, R. Oliveira, M. Luizelli, L. Gaspary, and M. Barcellos, "Survivor: an enhanced controller placement strategy for improving SDN survivability," in *GLOBECOM*, 2014.

[14] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "CAP for networks," in *HotSDN*. ACM, 2013.

[15] OpenDaylight, "OpenDaylight. http://www.opendaylight.org."

[16] P. Berde et al., "ONOS: Towards an open, distributed SDN OS," in *HotSDN*. ACM, 2014.

[17] S. Jain et al., "B4: experience with a globally-deployed software defined wan," in *ACM SIGCOMM*, 2013.

[18] Open Networking Foundation, "OpenFlow switch specification, version 1.4.0. http://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf."

[19] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *ACM SIGCOMM*, 2010.

[20] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *NSDI*, 2013.

[21] D. Kreutz, F. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, Jan 2015.

[22] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Hot-ICE*. USENIX, 2012.

[23] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *HotNets*. ACM, 2010.

[24] POX, "POX. http://www.noxrepo.org/pox/about-pox/."

[25] R. Wang, D. Butnariu, and J. Rexford, "Openflow-based server load balancing gone wild," in *Hot-ICE*. USENIX, 2011.

[26] V. Jeyakumar et al., "Millions of little minions: Using packets for low latency network programming and visibility," in *ACM SIGCOMM*, 2014.