RELIABILITY MANAGEMENT FRAMEWORK FOR SOFTWARIZED NETWORKS

A DISSERTATION
IN

Computer Networking and Communication Systems
And
Electrical and Computer Engineering

Presented to the faculty of
the University of Missouri-Kansas City
in partial fulfillment of
the requirements for the degree

DOCTOR OF PHILOSOPHY

by
HAIHONG ZHU

M.S., University of Virginia, 2001

B.S., Shanghai Jiao Tong University, 1993

Kansas City, Missouri
2021

RELIABILITY MANAGEMENT FRAMEWORK FOR SOFTWARIZED NETWORKS

Haihong Zhu, Candidate for the Doctor of Philosophy Degree

University of Missouri–Kansas City, 2021

ABSTRACT

The Software-Defined Networking (SDN) technologies promise to enhance the performance, reliability, and cost of managing both wired and wireless network infrastructures, functions, controls, and services (i.e., Internet of Things). However, centralized reliability management in Softwareization architecture poses both scalability and latency challenges. Significantly, the current OpenFlow Discovery Protocol (OFDP) in SDN induces substantial scalability, accuracy, and latency hurdles due to its gossipy, centralized, periodic, and tardy protocol.

This dissertation proposes a novel reliability management framework, which efficiently orchestrates different reliability monitoring mechanisms over SDN networks and synchronizes the control messages among various applications. The proposed framework facilitates multiple discovery frequency timers for each target over different stratum instead of using a uniform discovery timer for the entire network. It supports many common reliability monitoring factors for registered applications by analyzing offline and online network architecture information such as network topologies, traffic flows, virtualization architectures, and protocols. The framework consists of a high availability registration platform (HARP) and the topology-aware reliability management (TARman) and Bug Detection, Debugging, and Isolation (BuDDI) protocol facilities. The reliability management framework is implemented on both Ryu and Cisco's OpenDayLight (ODL) controllers. Extensive Mininet experimental results validate that framework significantly improves discovery message efficiency and makes the control traffic less bursty than OFDP with a uniform timer. It also reduces the network status discovery delay without increasing the control overhead. Our reliability management framework also proposes a novel network reliability cost model to ensure that the SLA covers customer service impact and damage. We classify network outages and calculate their effect on the network services to formulate a cost-based model. Besides, we have performed evaluations using various campus network outage scenarios. The proposed cost-based model enables customers to identify the service impact of unplanned network outages to their networks instead of entirely depending on the service provider's data.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Graduate Studies, have examined a dissertation titled “Reliability Management Framework for Softwarized Networks” presented by Haihong Zhu, candidate for the Doctor of Philosophy degree, and certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Sejun Song, Ph.D., Committee Chair
Department of Computer Science & Electrical Engineering

Cory Beard, Ph.D.
Department of Computer Science & Electrical Engineering

Lein Harn, Ph.D.
Department of Computer Science & Electrical Engineering

Ahmed Hassan, Ph.D.
Department of Computer Science & Electrical Engineering

Baek-Young Choi, Ph.D.
Department of Computer Science & Electrical Engineering

CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Dr. Sejun Song and co-discipline chair, Dr. Cory Beard for all their guidance, advice, and support throughout my doctoral research. Their great advice and guidance for Ph.D. study contributed to my growth in research skills about reading papers critically, discovering ideas, building up projects, and ultimately leading and managing projects throughout all research.

I sincerely thank all of my committee members, Dr. Baek-Young Choi, Dr. Lein Harn, and Dr. Ahmed Hassan for all their help and sincere advices when I approached them with questions. Their comments have helped to clarify and improve this work a lot. I also would like to thank all the classmates, faculty, staff, colleagues and friends because even a short chat and a small exchange of smiles enlightened my day.

Lastly, but most importantly, I'm grateful to my family for making this possible. My family have always been my happiness and driving force during my doctoral research.

CHAPTER 1

INTRODUCTION

The modern computer networking is more complicated now than it ever has been. A proliferation of network-enabled devices and bandwidth-intensive applications lead massive growth of customer's demands on the higher bandwidth and better quality of the network. As the network progresses, it is getting more difficult to efficiently manage the network. Network volume and complexity come to the fore as the main reasons that hinder efficient network management. As the number of network devices in the network gets higher, operating expense (OPEX) of the network accordingly increases. In addition, as more network layers, various protocols, and multiple vendors are introduced in a given network, network operation and management gets even more inefficient and difficult.

In order to grapple with closed, inflexible, complex, error-prone, and hard-to-manage production network problems, Software-Defined Networking (SDN) [4, 57, 79] has been proposed by many network companies and researchers. Particularly, fueled by increasing data center networking and cloud computing, SDN has been building up significant momentum toward the production network deployment. This recently emerging concept of a network architecture supports the programmable control interfaces by separating and abstracting the control plane from the data plane. This centralized control plane, which is called an SDN controller, becomes the brain of the underlying network. SDN enables many features, such as traffic engineering and network virtualization, in the form of an application on top of the controller of the network. In addition, SDN allows for rapid and simplified network exploration that improves network reliability, manage- ability, and security. Therefore, the centralized controller plays a very important role in controlling and operating, for example imposing policies and making decisions for routing, underlying network switches. Logically, it lies between northbound applications and southbound network devices. The OpenFlow protocol [63], which is managed by Open Networking Foundation (ONF) [61] is the de facto protocol that is being used for communication between the controller and the underlying network devices (e.g., OpenFlow switches). Northbound APIs are used for communication between the controller and the applications. These applications are developed to fulfill a specific purpose. The controller is an agent that connects applications and network devices and manages flow control to enable intelligent networking based on the applications' needs. As mentioned, this centralized architecture takes all the intelligence from the underlying network switches and leaves a flow table in the network switches. When the network switches receive new packets, they will forward these packets to the controller to decide where to send them.

High Availability (HA) of a network control system is important in a real network operation. Thus, provisioning redundancies a priori, then detecting failures and invoking mitigation mechanisms are the necessary steps in action. In the traditional network, HA is solely limited to data paths so that the system maintains a certain level of availability such as Service-Level Agreement (SLA). In the SDN architecture, the issue of HA becomes more critical especially for the controllers of SDNs, as they are responsible for the functions of the network switches. Furthermore, the SDN architecture poses more complexity on the HA issues by creating a couple of separate HA network domains, in addition to the data plane, such as controller cluster networks as well as control and data plane connection networks. Although there have been a few recent studies that focus on the failures of switches or links connecting them in data plane, little work is found to consider the failures of controller cluster network itself nor to orchestrate the failure detection and recoveries of all the separate HA network domains. Another important aspect with regard to HA is fast and accurate failure detection. Detecting a failure quickly and accurately is also very critical to maintain HA of a system. This is because slow or wrong failure detection delays the root cause analysis of the problem and delays recovery ofthe system. As a result, overall performance of system's HA would be degraded. Therefore, we additionally focus on how fast we can detect failures in order to reduce the downtime of the network and improve HA of the network in the SDN environment.

Along with HA, scalability of a network system is also important in a real network operation. There has been a few research conducted on the scalability issue of an SDN controller. Those researches can be divided into three types such as improving the capacity of the controller itself by using parallel processing and multi-threads, devolving some control functions to the OpenFlow switches, and clustering multiple controllers. Most of the existing approaches try to increase the system capacity to accommodate the increasing number of network traffic between a controller and underlying switches. However, if we look at the network messages between a controller and underlying switches more closely, we can find each message has different importance according to the activities and status of the current network.

In this research, we study various aspects of controller's HA impacting the overall performance of SDN. We identify several critical HA issues and propose a solution for each problem. In addition, we also study and propose solutions for SDN scalability using prioritization and classification techniques.

1.1 Software-Defined Networks

SDN is an emerging computer networking paradigm that supports programmable interfaces providing an agile and convenient way to customize the network traffic control. The main contribution of the SDN architecture is the disaggregation of the vertically integrated networking planes in order to improve network flexibility and manageability. The control plane of the vertical network stack is decoupled and abstracted to interact and handle all of the underlying network devices within its domain. It is logically centralized and is called an SDN controller. An SDN controller can run on a commodity server. With this centralized controller in the network, it gets the global view on the underlying network with ease. In addition, new services can be loaded up to the controller and each differentiated service can utilize the global view of the controller to achieve its optimized performance. Therefore, SDN brings many benefits such as easier traffic optimization, agile new feature deployment, and reduced management cost.

As illustrated in Figure 1, this centralized architecture takes all the intelligence from the underlying network switches and leaves a flow table in the network switches. Therefore, when the network switches receive packets, they will search the matching rules from the flow tables. Each flow table consists of flow entries and there are six

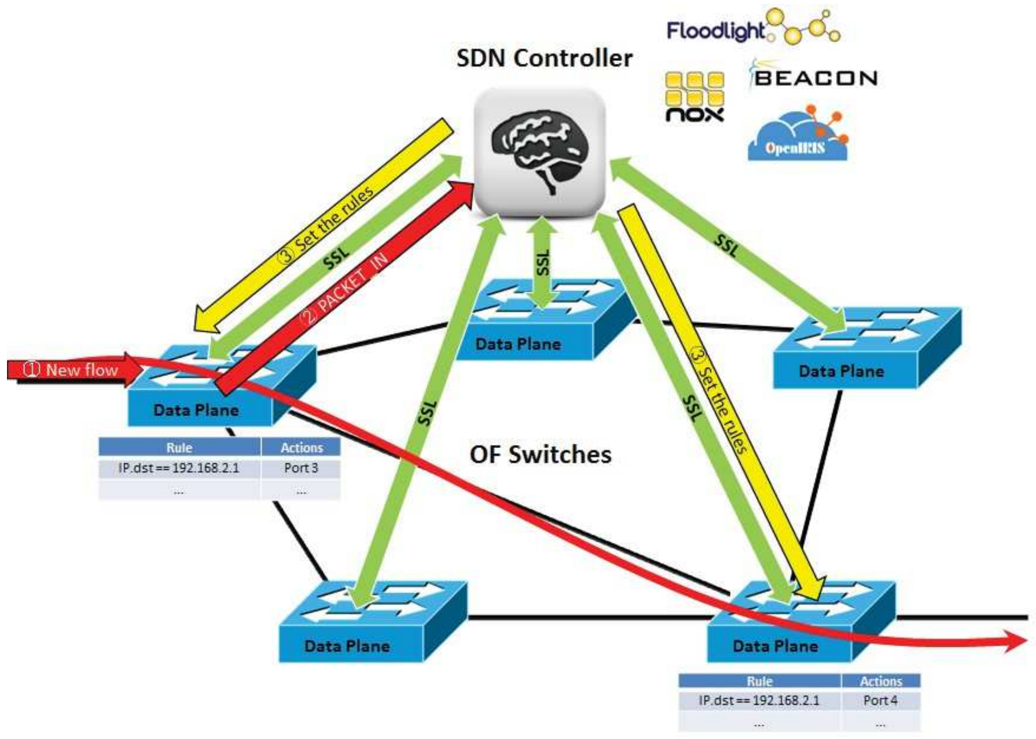


Figure 1: Software-Defined Networks (SDN) architecture

main components in each flow entry such as match fields, priority, counters, instructions, timeouts, and cookie. The match fields category, which consists of the ingress port and packet headers, is the most important factor to examine incoming packets. If there is a matching flow entry, the switch will handle the packets according to the associated action of its instructions category. If there is no matching flow entry (e.g., new packets), then they will forward these new packets to the controller in order to ask how to handle them

Table 1: Open Source SDN controllers

Controller	Languages	OpenFlow Protocol	Copyright
Beacon	Java	1.0	Apache 2.0 license
Floodlight	Java	1.0	Apache 2.0 license
IRIS	Java	1.0 ~ 1.3	Apache 2.0 license
Maestro	Java	1.0	GNU LGPL v2.1
Mul	C	1.0 ~ 1.4	GNU GPL v2.0
Nox	C++ & Python	1.0	Apache 2.0 license
OpenDaylight	Java	1.0 ~ 1.4	Eclipse Public License v1.0
Pox	Pyhon	1.0	Apache 2.0 license
Ryu	Python	1.0 ~ 1.4	Apache 2.0 license
Trema	Ruby & C	1.0	GNU GPL v2.0

Many companies, universities, and research institutes have been involved to develop and improve SDN for practical use. Major components of SDN are SDN controllers, OpenFlow protocol, and OpenFlow switches. Table 1 and 2 compares the specifi- cations of the current SDN controllers. Various SDN controllers have been developed and running commercially and academically such as Beacon [11], Floodlight [13], IRIS [14], Maestro [3], Mul [15], Nox [29], OpenDaylight [17], Pox [18], Ryu [19], Trema [20] and etc.

Being a detailed embodiment of SDN, OpenFlow [63] is a configuration language and protocol that abstracts the data plane of a networking device into a flow based architecture composed of a cascading set of classifiers and policy enforcement. The OpenFlow protocol is currently the de facto standard being used for the communication between an SDN controller and an OpenFlow switch. It is an open source project managed by Open Networking Foundation (ONF) [61]. It enables that an SDN controller controls the

Table 2: Commercial SDN controllers

Controller	Languages	OpenFlow Protocol	Company
Big Network Controller	Java	1.0 ~ 1.3	Big Switch Networks
ONE	Java	1.0 ~ 1.4	Cisco
Contrail	Java & Python	1.0 ~ 1.3	Juniper Networks
ProgammableFlow	Ruby & C	1.0 ~ 1.3	NEC
SDN VE	Java & Python	1.0 ~ 1.3	IBM
ViSION	Java	1.0 ~ 1.3	HP
Vyatta	Java	1.0 ~ 1.4	Brocade

forwarding plane of switches/routers. It also helps an SDN controller collect statistic information from the network in order to have a global view on the underlying network. The OpenFlow protocol is currently being implemented by major switch/router vendors to support and deliver OpenFlow-enabled products. Various OpenFlow switches are commercially available in the market.

1.2 Network Availability

Achieving network availability is one of the most important operational objectives of network service providers (NSPs). Availability is the fraction of a time that a system or component is continuously operational. Figure 2 describes terminologies related to network availability. HA can be measured by three main metrics such as *Mean Time Between Failures* (*MTBF*), *Mean Time To Repair* (*MTTRr*), and *Mean Time To Failure* (*MTTF*). *MTBF* is an expected average time between failures of a network component. *MTTRr* is an expected average time to restore a failed network component. The average downtime can be further divided into two parts such as *MTTD* and *MTTRc*. *MTTD* is an

Table 3: Network downtime according to the number of nines

Availability	Downtime per year	Downtime per month	Downtime per week
90%	36.5 days	72 hours	16.8 hours
99%	3.65 days	7.2 hours	1.68 hours
99.9%	8.76 hours	43.8 minutes	10.1 minutes
99.99%	52.56 minutes	4.32 minutes	1.01 minutes
99.999%	5.26 minutes	25.9 seconds	6.05 seconds

expected average time to detect a failed network component and *MTTRc* is an expected average time to recover the failed network component. Lastly, *MTTF* is a mean time to failure once the network component starts working normally. Therefore, the availability of the network component can be calculated by the formula as shown in Equation 1.1. Table 3 shows the operational performance (i.e., downtime) according to the number of nines. As we have more nines, we can say that the network is more stable.

$$Availability = \frac{MTTF}{MTBF} = \frac{MTBF - MTTD - MTTRc}{MTBF} \tag{1.1}$$

Many technologies have been developed to increase availability and ensure the reliability requirements. As shown in Table 5, traditional HA architecture supports link bundling, multipath routing, system redundancy mechanisms along with efficient state synchronization, and failure detection and handling protocols. These HA mechanisms are implemented in each network device as a distributed protocol to handle the network problems according to the dedicated network topologies.

The emerging concept of SDN decouples the control plane from the underlying network devices and abstracts it out as a centralized service. Many NSPs are very supportive on its deployment due to potential benefits such as operational cost reduction and

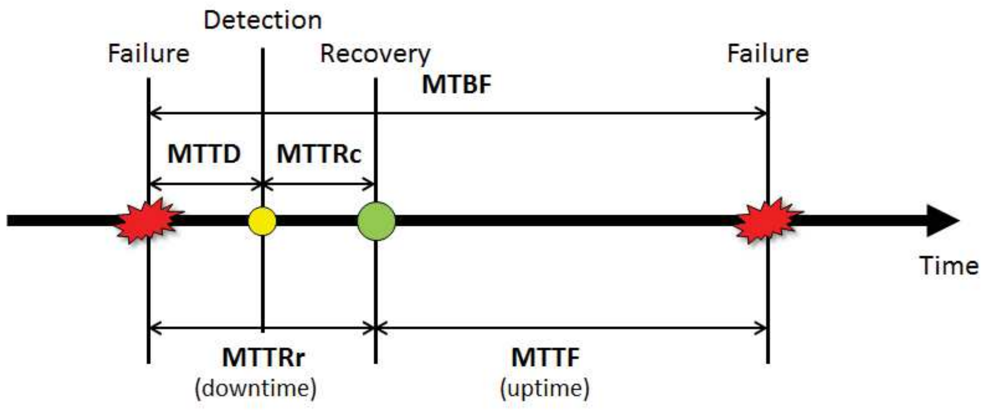


Figure 2: Network availability timeline

enhanced system resilience. However, unlike traditional networks, the existing HA mechanisms may face many critical challenges to achieve the same Service Level Agreement (SLA) of HA for the network services in the SDN environment where the out-of-band connections and controller connections (*control path*) exist between the control and data planes and between controllers, respectively. The issue of HA becomes more crucial on the 'controllers' than the 'OpenFlow switches', as they are responsible for the intelligent decision of the OpenFlow switch policies.

1.2.1 Issues of Existing High Availability Solutions

Although there have been a few recent studies that focus on the failures of OpenFlow switches or links connecting them in the data plane [23] and the controller cluster networks for improving both scalability and reliability [50, 75], respectively, little work is found to sophisticatedly exploit HA mechanisms over the abstraction domain. For example, even though ONIX [50], HyperFlow [75], and ONOS [1] consider reliability of

Table 4: High availability mechanisms

Mechanisms	Protocols
Link bundling	Link Aggregation Control Protocol (LACP) [36], EtherChannel [33]
Multipath routing	Equal-Cost Multi-Path routing (ECMP) [31]
System redundancy	Virtual Router Redundancy Protocol (VRRP) [41], Host Standby Router Protocol (HSRP) [35], Resilient Packet Ring (RPR) [39]
State synchronization	Non-Stop Routing (NSR) [38], Non-Stop Forwarding (NSF) [37], Stateful Switch-Over (SSO) [40]
Failure detection and handling	Ethernet Automatic Protection Switching (EAPS) [30], Ethernet Ring Protection Switching (ERPS) [32], Fast Re-Routing (FRR) [34]

the controller cluster via the distributed physical controllers and synchronization among controllers, they don't extensively exploit reliability problems caused by the limitations of the current OpenFlow specification [64].

To address this problem, we propose and implement three HA strategies toward the *control path* of the SDN environment such as *coupling logical connections with physical redundancy*, *virtualizing the controller cluster*, and *fast and accurate failure* *detection and recovery.*

1.3 Network Scalability

In the traditional network systems, the main network functionalities such as data, control, and management planes are distributed and embedded within the vendor spe- cific networking devices and are managed remotely by EMSs [26], NMSs [60], OSSs, and BSSs [65] through provisioning and configuration. As the network systems become bigger, faster, and more complex over multiple administration domains and diverse com- ponents, they need to handle multiple protocols with cross-layer interactions, support various applications and services over multi-tenant policies, and are managed over uncertain underlying topology and internal structure. At the same time, the network services and applications are expected to be deployed quickly and more dynamically on the large-scale networking systems while insuring security, reliability, performance, traffic isolation, end-to-end virtualization and priority treatment. However, providing quick and easy dynamic network adaptability is intrinsically difficult problems for the legacy network systems, as they barely cope with the complexity through the layers of the extensive and expensive remote provisioning and configuration.

More specifically, traffic and resource monitoring is the essential function for large-scale enterprises, service providers, and network operators to ensure reliability, availability, and security of their resources. For this reason, many large-scale enterprises and providers have been investing in various stand-alone dedicated monitoring solutions. However, they find that a proprietary and dedicated stand-alone hardware-based appliance per-feature is inflexible, slow to install, and difficult to maintain as well as the cost is prohibitive. To cope with such a huge required investment, many enterprises are looking for outsourcing alternatives and providers are also looking for means to reduce the cost.

As networks are evolving towards software defined networks, control and management functions are logically centralized and real-time, scalable, and dynamic monitoring of managed resources is a key to make precise control decisions. In addition to this, virtualization (e.g., Network Virtualization (NV) and Network Function Virtualization (NFV) [59]) of the underlying computing, network resources including Layer 4 (transport) ~ Layer 7 (application) capabilities, and network services has emerged as a complementary approach along with SDN. Specially, NFV provides more flexible and programmable monitoring functions which are used to be built in specific hardware. To provide such flexible and programmable monitoring functions, virtualization ofmonitoring function itselfcan be a solution. That is, a monitoring function of a particular objective can be instantiated on demand in real-time whenever a new monitoring requirement occurs and can dynamically be deleted once its demand completes. Since main benefit of network function virtualization is the chaining of its functionality, virtual monitoring function can be utilized as a part of such a virtual function chaining. Despite SDN and NFV promises flexibility, simplicity, and cost-effectiveness, the abstractions toward the remote and centralized control and management tend to introduce the following challenging problems:

- *Scalability problem*: The proposed SDN architecture tends to open up control messages between the controllers and the forwarding devices to the communication networks, which is spatiotemporally concentrated around the centralized controller. As studied in DevoFlow [22], DIFANE [81], ONIX [50], SDN imposes excessive control traffic overheads in order for the controller to acquire the global network visibility. More significantly, the overheads will be further increased by the traditional network management events as well as the application specific control traffic, as they may use the same physical network paths, buffers, and I/O channels at the same time. If the overheads are not controlled properly, they can cause various scalability problems on the networking devices, controllers, and the network itself including slow message processing, potential message drop, delayed root cause analysis, and late response against urgent problems.

- *Inaccurate and unreliable management problem*: In traditional network systems, the network management practice mainly takes remote approaches coping with the network-centric infrastructure. However, since the network events occurred within the network should be inferred by the remote management systems, the potential network problems are often accumulated and enlarged, and the diagnosis is delayed, inaccurate, unreliable, and not scalable. SDN's remote and centralized control tends to extend the legacy network management problems into the control plane.

- *Multiple management pillar problem*: Although SDN's management plane is the relatively unexplored area, either the SDN controller agnostic application of the

incumbent management protocols or the full integration of the management plane into the controller protocols such as OpenFlow cannot be the viable approach for the highly dynamic SDN management. Moreover, there are growing expectations for the fine grained management of the customer specific services and applications. Many existing SDN approaches evidence that SDN allows variety of heterogeneous application interfaces and protocols to the data plane. For example, according to the most recent OpenDaylight controller architecture, the SDN control and management provide an expendable facility for the protocol specific management. Al- though OpenFlow gained its visibility as the most fundamental pillar to support SDN, it is actually one of the many programming interfaces to the data plane. Multiple other interfaces and protocols such as OVSDB [66], SNMP [72], and various NFV applications also play a significant role in the evolution of the SDN management architecture.

- *Heterogeneous deployment problem*: Moreover, in the practical network operation point of view, the SDN deployment may take the gradual transition instead of the one-night-all transition. Therefore, integrating existing services and protocols with SDN is an essential step for the transition. Also, some of the network systems may want to employee the SDN functions only partially. For example, inter data center networks may want to use the provisioning functionality only, but to use their existing management tools. This will result a heterogeneous management environment. A complex combination of multiple and heterogeneous management channels introduces a significant scalability problem.

We have investigated various network service aspects including agility, accuracy, reliability, and scalability in order to identify an effective SDN network management system. This dissertation intensively focuses on scalability issues. We propose a filtering and common processing module that facilitates various communication interfaces to collect network events. It also provides common filtering and event mitigation functions to simplify the event processing for the user-defined monitoring modules. In order to validate the functionalities of our proposed schemes, we implemented the proposed schemes and

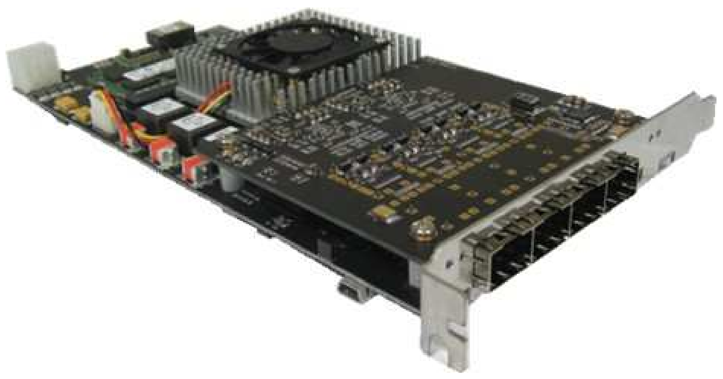
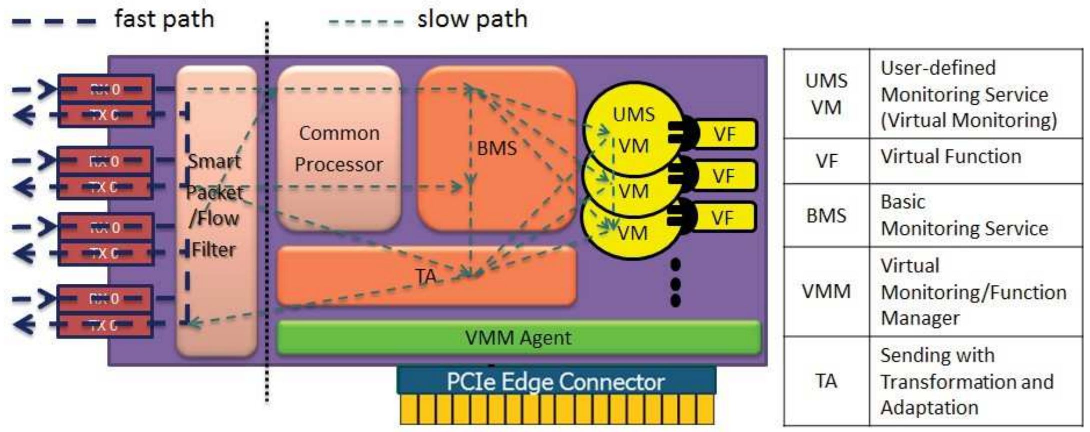


Figure 3: Software-defined Unified Monitoring Agent (SUMA) board (MDS-40G)

metrics in OpenFlow with OpenWrt [54] based switches. In collaboration with Electronics and Telecommunications Research Institute (ETRI), we also implemented our pro- posed modules in an intelligent management middlebox called Software-defined Unified Monitoring Agent (SUMA) [8] that becomes one logical point of intelligence for the integrated management services. SUMA is an essential switch-side middlebox that provides control and management abstraction and filtering layer among vNMS, SDN controllers, legacy NMS, and Openflow switches. SUMA performs a light weight event detection and filtering and the correlation will be conducted in vNMS. The two-tier framework is used to balance the performance impact between network devices and controllers, to provide scalability, and to ensure dynamic deployment.

1.3.1 Software-defined Unified Monitoring Agent

Figure 3 shows the SUMA board. SUMA is implemented in a couple of multi-core network processing cards powered by a Tile-Gx36 [74] processor. It supports 36



UMS VM	User-defined Monitoring Service (Virtual Monitoring)
VF	Virtual Function
BMS	Basic Monitoring Service
VMM	Virtual Monitoring/Function Manager
TA	Sending with Transformation and Adaptation

Figure 4: SUMA implementation structure

cores and each core has 1.2GHz clock speed. It also supports 10 Gbps packet processing capacity. Each card supports four 10G ports. As illustrated in Figure 4, the smart packet and flow filter take the fast-path to process the incoming packets with the line-rate. Common processing, basic monitoring functions are all processed over the slow-path. Virtual monitoring and function manager are implemented in the host user space and interact with other functions via the virtual monitoring manager agent in the card. Our solutions for scalability issues are implemented as a User-defined Monitoring (UM) service and are realized on a Virtual Machine (VM).

1.3.2 Issues of Existing Scalability Solutions

In order to overcome SDN's scalability issues, there has been a few research conducted. These solutions can be categorized into three types. As shown in Figure 5, the first type of the solutions tries to improve the capability of the controller itself by using

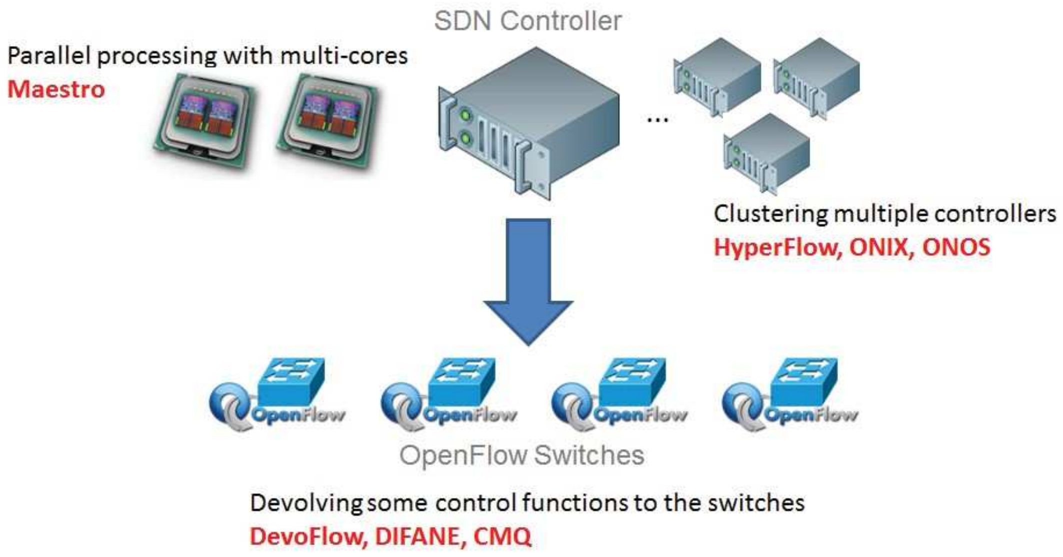

SDN Controller
Parallel processing with multi-cores
Maestro
...
Clustering multiple controllers
HyperFlow, ONIX, ONOS
OpenFlow
OpenFlow Switches
Devolving some control functions to the switches
DevoFlow, DIFANE, CMQ

Figure 5: Existing solutions to improve scalability

parallel processing and multi-threads [20]. The second type of the solutions is devolving some control functions of the SDN controller to the OpenFlow switches. DIFANE [81] runs the partitioning algorithm that divides the rules evenly and devolves those partitioned rules across *authority switches*. These *authority switches* will handle new flows instead of the SDN controller. DevoFlow [22] mainly reduces the interactions between OpenFlow switches and the SDN controller using rule aggregation, selective local action, and approximating techniques. Another devolving solution, called Control-Message Quenching (CMQ), sends only one packet-in message for each source-destination pair, suppresses and enqueues the following un-matching packets until the switch receives packet-out or flow-mod message and installs the matching rule in its flow table. It reduces unnecessary packet-in messages from the OpenFlow switch to the controller. The third type of the solutions is clustering multiple physical controllers. In HyperFlow [75], the authors tried to provide scalability, using as many SDN controllers as necessary and keep the global visibility of link state changes. However, it has lower visibility for the flowlevel statistics. ONIX [50] is also distributed control platform for large-scale networks. It provides more general APIs than earlier systems, so it is easier to distribute controllers andcollect fine-grained flow statistics from the distributed controllers. ONOS [1] is an experimental open source distributed SDN operating system which provides scalability for SDN control plane and achieves strong consistency of the global network visibility.

Although the existing scalability solutions are to reduce the amount of the control messages within one controller instance, they are not able to reduce the overall quantity of the control messages from the network. In addition, they don't sufficiently consider the quality of individual controller messages, especially, while the control messages are competing the same resources against themselves or other messages. To address this problem, we propose and implement a two-tier network management framework in two different approaches; embedded approach and agent-based approach. Each approach includes various schemes and algorithms using event filtering, annotation, prioritization and classification techniques to alleviate the workloads of SDN controllers.

1.4 Objectives of the Dissertation

The objectives of the dissertation are to suggest new ways to remove or alleviate problems of the existing solutions and current specification; and develop management frameworks that improve HA and scalability of the current Software-Defined Networking

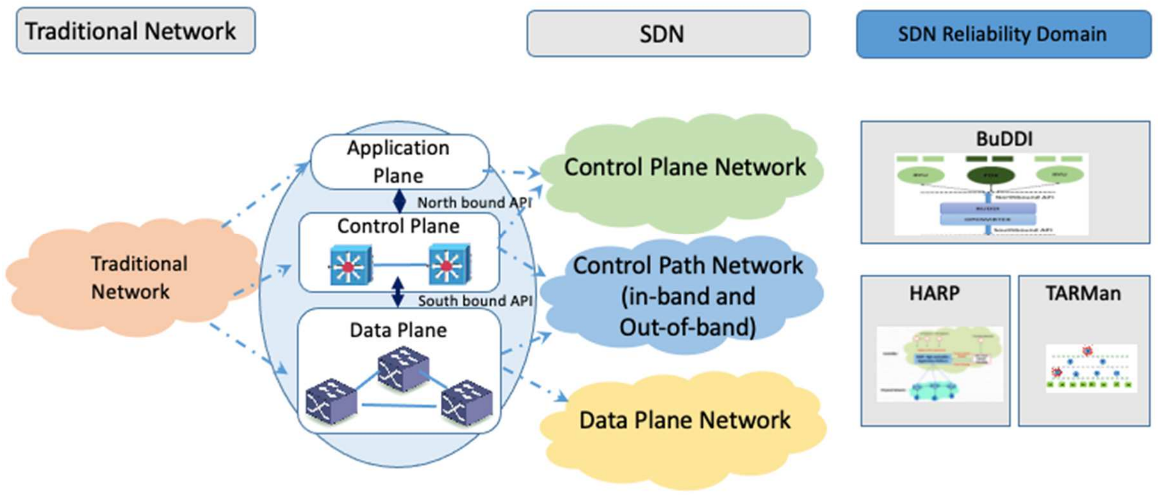


Figure 6: Overall SDN Reliability Framework Architecture

systems. As illustrated in Figure 6, the proposed framework handles different issues (e.g., HA and scalability).

1.5 Scope and Contribution of the Dissertation

In this dissertation, we focus on two aspects of SDN management such as HA and scalability. The main contributions of this dissertation are as follows.

- Online software bug detection, debugging, and isolation (BuDDI) middlebox architecture is proposed to solve for software-defined network controllers Common Software Failure problem.
- HARP (High Availability Registration Platform) orchestrates control messages over the network virtualization components.
- TARMan (Topology Aware Reliability Management) promotes an Architectures

Awareness in reliability management. By individualizing the inspection timers, TARMan can detect outage faster with less control messages.

- We also propose distinctive network reliability cost model to ensure that the SLA covers customer service impact and damage. We classify network outages and calculate their effect on the network services to formulate a cost-based model.

1.6 Organization

The rest of this dissertation is organized as follows. In Chapter 2, we review related work dealing with the HA and scalability issues of SDN. Before we dive into the HA and scalability issues of SDN, we discuss the traditional network availability in Chapter 3. In Chapters 4 and 5 we identify new problems of SDN in regards to reliability, HA and scalability issues and propose their practical solutions. We address the novel reliability cost model and associated Markov model in Chapter 6. Finally, Chapter 7 summarizes and concludes this dissertation and discusses future research goals.

CHAPTER 2

RELATED WORK

As the control plane in SDN is separated from the data plane and becomes a remote and centralized controller of the network, two major operational problems have arisen. First of all, HA issues of the SDN controller becomes very critical. One centralized controller of the network means a single point of failure. Since the controller is the brain of the network, the network could be disrupted without a proper operation of the controller. In addition, since the underlying forwarding devices (e.g., OpenFlow switches) don't have its own decision engine, it may induce delay to recover from switch failures (e.g., hardware and software) compared to the legacy network. Therefore, it is also critical to have a fast recovery mechanism to improve HA of the data plane. These concerns on HA issues motivate several research on HA in SDN. We will discuss them in detail in the following sections. Secondly, as the number of underlying network devices, protocols, and applications running on top of the SDN controller drastically increases, the capacity of the controller could not be enough to handle all the requests from the network, protocols, and applications. In addition, an OpenFlow switch also could be under congestion when it receives more packets than its peak capability or is under malicious attacks. These issues motivate researchers to study on scalability of the SDN controller. Therefore, in order to provide a highly reliable and robust SDN environment, we have to deeply consider these two major properties of the SDN controller.

In terms of HA in SDN, two types of issues have been studied so far. One is data plane HA and the other is control plane HA. Data plane HA of SDN can be further categorized into two topics such as fast failure detection on a data plane and HA for application servers that are running in the SDN environments. The scheme for the fast failure detection on a data plane utilizes the OpenFlow switch's link signals to check connectivity among neighboring switches or delegates fault management to the OpenFlow switches by extending the OpenFlow protocol to support the monitoring function. The scheme for the server HA mainly focuses on HA between OpenFlow switches and multiple server replicas [23, 47, 49, 52]. In addition to the above proposed HA strategies, it is also significant to detect failures in the network fast and accurately so the network can be recovered in a timely manner so as to maintain a highly available system [78]. There are a few work done for fast failure detection in the SDN area. The existing research focused on a data plane network. To compare with the existing research work, our research direction is unique, in that it mainly focuses on the HA issue of controller networks and a network between a controller and OpenFlow switches.

As previously mentioned, the separation of the control plane from the data plane introduces a centralized SDN controller. Since the SDN controller administers the underlying network and manages all the flows over the underlying network devices, it is easy to expect that it may have an intrinsic scalability issue on the SDN controller. Along with the HA research work, there has been a few research conducted on the scalability issue of the SDN controller. Those research can be divided into three types. The first type of

Table 5: High availability mechanisms

Mechanisms	Protocols
Link bundling	Link Aggregation Control Protocol (LACP) [36], EtherChannel [33]
Multipath routing	Equal-Cost Multi-Path routing (ECMP) [31]
System redundancy	Virtual Router Redundancy Protocol (VRRP) [41], Host Standby Router Protocol (HSRP) [35], Re-silient Packet Ring (RPR) [39]
State synchronization	Non-Stop Routing (NSR) [38], Non-Stop For-warding (NSF) [37], Stateful Switch-Over (SSO) [40]
Failure detection and handling	Ethernet Automatic Protection Switching (EAPS) [30], Ethernet Ring Protection Switching (ERPS) [32], Fast Re-Routing (FRR) [34]

the solutions is dedicated to improving the capacity of the controller itself by using multi-cores with parallel processing and multi-threads [3]. The second type of the solutions is devolving some control functions to the OpenFlow switches [22,47,56,81]. These hybrid approaches allow some degrees of intelligence to the OpenFlow switches. By offloading some control functions from the controller, they expect to reduce workloads imposed on the controller. We will see in detail what functions/intelligence are left in the Open- Flow switches. Last but not the least, the third type of the solutions is clustering multiple controllers [1, 50, 51, 76]. These approaches show how they can synchronize the global visibility of network state changes across the multiple controllers in the cluster.

2.1 Related Work of Traditional Availability Issues

HA is a well-established research topic and many technologies have been developed to increase availability and ensure the reliability requirements. As shown in Table 5, traditional HA architecture supports link bundling, multipath routing, system redundancy mechanisms along with efficient state synchronization, and failure detection and handling protocols. These HA mechanisms are implemented in each network device as a distributed protocol to handle the network problems according to the dedicated network topologies. In addition, most of the implementations of these HA mechanisms are proprietary. Therefore, they are not readily available in the SDN environment. LACP [36] and VRRP [41] can be easily adopted to the SDN system. However, they don't cover the synchronization between the SDN controllers, correlation of failures between the control plane and the data plane, and inter connection HA, we need a sophisticated HA mechanisms specifically designed for SDN.

2.2 Related Work of High Availability Issues in SDN

In Section 2.1, we have presented types of HA techniques and discussed the current HA mechanisms in the traditional network. HA is a well-known research topic and well-established. However, these HA features don't fully consider the correlation between failures of the control plane network and inter-connection network that are newly introduced in SDN. There have been a few recent studies that focus on the failures of OpenFlow switches or links connecting them to facilitate the HA feature of the data plane in SDN. In this section, we categorize HA issues in SDN into two topics; data plane HA

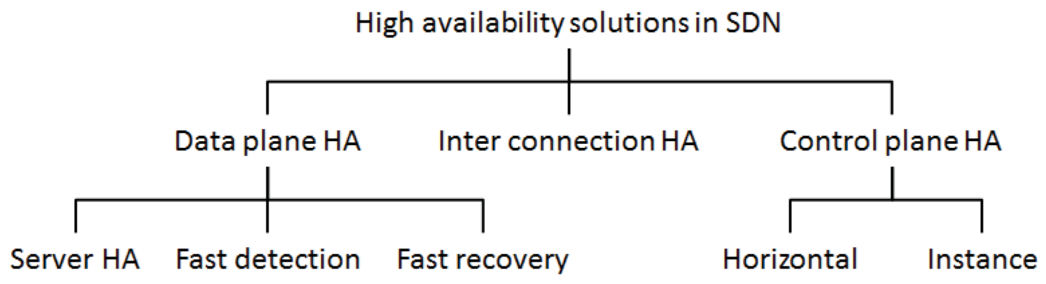


Figure 7: High availability classification in SDN

and control plane HA and talk about the current research work. Table 6 presents a comprehensive view of the current high availability research in SDN. The details are explained in the following sections.

2.2.1 Data Plane High Availability

As we mentioned, data plane HA in SDN can be further categorized into two topics; application server HA and fast failure detection. First of all, we discuss the current research work related to the application server HA. The study on application server HA in SDN can be found in [78]. The authors proposed RuleBricks that provides HA in existing OpenFlow policies. It primarily focuses on embedding HA policies into OpenFlow's forwarding rules. They implemented RuleBricks by utilizing an expressive brick-based data structure instead of naive tree-based data structure. They show that RuleBricks maintains linear scalability with the number of replicas (i,e,. *backup rules*) and offers approximately 50% reduction in the *active ruleset*.

Now, we discuss the current research work in the area of fast failure detection and

Table 6: High availability research: comprehensive view

Ref.	Data plane HA			Inter connection HA	Control plane HA	
	Server HA	F.D.	F.R.		Horizontal	Instance
[78]	✓					
[23]		✓				
[47]		✓	✓			
[49]			✓			
[42]					✓	
[76]					✓	
[50]					✓	
[1]					✓	
[3]						✓

F.D.=Fast detection, F.R.= Fast recovery

recovery on the data plane. As we discussed in the definition of network availability, it is very important to quickly detect failures in the network as well as to recover the network as soon as possible after the failure detection. Failure detection and network recovery in a timely manner maintain highly available system. There have been a few studies conducted on fast failure detection and recovery in SDN and most of them have focused on the data plane network. Desai et al. [23] proposed an algorithm that utilizes the OpenFlow switch's link signal to check the connectivity among neighboring switches for fast failure detection. This scheme notifies the link failure to all the neighboring switches in order to refrain from sending messages in the direction of the failed link so it can minimize unnecessary traffic in the network and reduce the effects of the link failures. Their algorithm enables failure detection faster than the controller which identifies the failed link through heartbeat messages and sends out an update. However, their algorithm does not contribute to the recovery of the network. Kempf et al. [47] also considers fast failure detection and recovery by extending the OpenFlow protocol to support a monitoring function on OpenFlow switches. They followed the fault management operation of MPLS-TP for the implementation and achieved fault recovery in the data plane within 50 ms. Kim et al. [49] proposed an SDN fault-tolerant system, named CORONET (controller based robust network), that mainly focuses on recovering the data plane network from multiple link failures. Their proposed modules can be implemented and integrated into NOX con- troller. They summarized challenges on building a fault-tolerant system based on SDN but they didn't describe the proposed modules in detail.

2.2.2 Control Plane High Availability

Along with data plane HA, control plane HA has also been studied for various aspects. Hellen et al. [42] discussed about controller's physical placement in the network. They tried to optimize the number of controllers and their location in the network. By connecting an OpenFlow switch to the closest controller in the network, it can re- duce control delay and contribute to improvement of high availability. Tootoonchian et al. [76], Koponen et al. [50], and Berde et al. [1] proposed HyperFlow, ONIX, and ONOS, respectively. These proposed frameworks establish one logical controller consisting of physically distributed controllers in the cluster. Since they run on the multiple physical controllers, the slave controllers can operate the network when the master controller goes down. Even though HyperFlow, ONIX, and ONOS consider some aspects of reliability of the controller cluster via the distributed physical controllers, their main concerns are

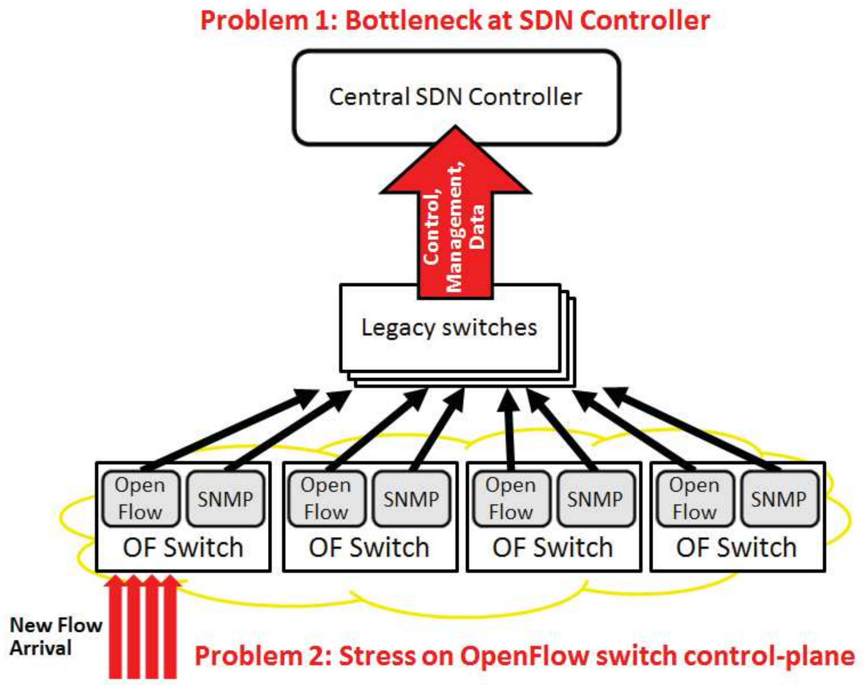


Figure 8: Two major scalability issues in SDN

scalability and synchronization of network status among multiple physical controllers. Cai et al. [3] proposed Maestro controller which supports software HA. A task manager of Maestro manages incoming computations and evenly distributes work to each SDN controller instance at each core of the processor. Since it exploits multi-core architecture, it can re-distribute the work evenly at the time of the core crash or software crash.

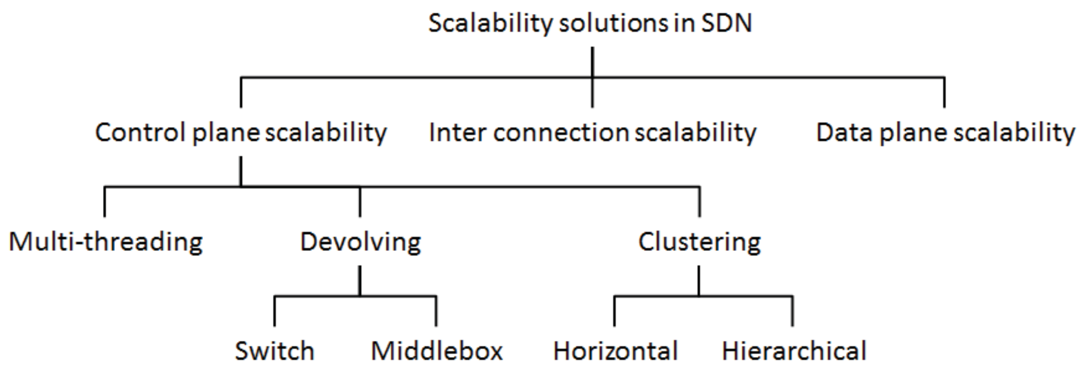


Figure 9: Scalability classification in SDN

2.3 Related Work of Scalability Issues in SDN

Along with the HA research work, there has been a few research conducted on scalability issues in SDN. Figure 8 illustrates the scalability problems that can be addressed in the current SDN architecture. The first problem can be seen in the controller. As introduced, SDN relies on a centralized controller to operate the underlying network and opens up control messages to communicate between the controller and the forwarding devices. As the size of the underlying network gets bigger, the capacity of the SDN controller could be quickly saturated. The second problem can be observed in the OpenFlow switch. Unlike the traditional network, the forwarding device in SDN has to communicate with the SDN controller to make a decision for forwarding or routing and get network policies. Specially, it is a mandatory procedure for an OpenFlow switch to send new flow packets to the SDN controller in an encrypted format such as the packet-in message to cope with them. This could be extra workloads and saturate the OpenFlow switch.

Current scalability research in SDN primarily focuses on the controller part. Those researches can be divided into three types such as improving the capacity of the controller itself by using parallel processing and multi-threads, devolving some control functions to the OpenFlow switches, and clustering multiple controllers. Table 7 presents a comprehensive view of the current scalability research in SDN. The details are explained in the following sections.

2.3.1 Controller Enhancement with Multi-threading

The first type of the solutions tries to improve the capacity of the controller itself. Cai et al. [3] proposed the Maestro controller for scalable OpenFlow network control. Since the SDN controller is the only brain of the network that copes with all the requests from the underlying network devices, it could be a performance bottleneck of the network system. The authors exploits parallelism to improve the capacity of the controller. They implemented Maestro that can support multi-core processors with parallel processing and multi-threads. Their experiments show that the throughput of Maestro can achieve near linear scalability on a multi-core processor server.

2.3.2 Devolving Control Functions

The second type of the solutions is devolving some control functions to the Open-Flow switches. There are two well-known papers such as DIFANE [81] and DevoFlow [22]. DIFANE runs a partitioning algorithm that divides the rules evenly and devolves those partitioned rules across *authority switches*. These *authority switches* will handle new flows instead of the controller. DevoFlow mainly reduces the interactions between OpenFlow switches and the SDN controller using filtering and sampling such as rule aggregation, selective local action, and approximating techniques. Another devolving solution, called Control-Message Quenching (CMQ), is proposed by Luo et al. [56]. The switch with CMQ sends only one packet-in message for each source-destination pair, suppresses and enqueues the following un-matching packets until the switch receives packet-out or flow-mod message and installs the matching rule in its flow table. It reduces unnecessary packet-in messages from the OpenFlow switch to the controller. Lastly, the work done by Kempf et al. [47] also can be considered one of the devolving schemes. The authors claimed that the centralized fault management has serious scalability limitations. Therefore, the proposed scheme delegated fault management to the OpenFlow switches by extending the OpenFlow protocol to support the monitoring function.

2.3.3 Clustering Multiple Controllers

The last type of the solutions is clustering physically distributed controllers into a logically centralized controller in order to increase the capacity of the controller. There are two types of clustering techniques such as horizontal clustering and hierarchical clustering. In the horizontal clustering, each controller plays a role of master or slave. They could have the same functionalities or may have different functionalities based on the configuration and implementation. In HyperFlow [76], the authors tried to provide scalability, using as many controllers as necessary and keep the global visibility of link state changes. However, it has lower visibility for the flow-level statistics. Koponen et al. [50]

Table 7: Scalability research: comprehensive view

Ref.	Focus		Scalability method			Devolving		Cluster	
	Cont.	Switch	M.C.	Devol.	Cluster	Switch	Other	Horizontal	Hier.
[3]	✓		✓						
[81]	✓			✓			✓		
[22]	✓			✓		✓			
[56]	✓			✓		✓			
[47]	✓			✓		✓			
[76]	✓				✓			✓	
[50]	✓				✓			✓	
[1]	✓				✓			✓	
[51]	✓				✓			✓	
[80]	✓				✓				✓
[70]	✓				✓				✓
[53]	✓				✓				✓

Cont.= Controller, M.C.= Multi-cores with multi-threads, Devol.= Devolving,Hier.= Hierarchical

proposed ONIX which is also distributed control platform for large-scale networks. And it provides more general APIs than earlier systems, so it is easier to distribute controllers and collect fine-grained flow statistics with the distributed controllers. Berde et al. [1] proposed ONOS. It is an experimental open source distributed SDN operating system which provides scalability for SDN control plane and achieves strong consistency of the global network visibility. Krishnamurthy et al. [51] tried to improve performance of the current distributed SDN control platforms by proposing a novel approach for assigning SDN switches and partitions of SDN application state to distributed controller instances.

The authors focused on two metrics such as minimizing flow setup latency and minimizing controller operating costs. Their scheme shows that a 44% decrease in flow setup latency and a 42% reduction in controller operating costs.

The second clustering technique uses the hierarchical structure. Controllers in the cluster can be classified into two types of controllers; a super controller and a regular controller. Yeganeh et al. [80] proposed an efficient and scalable framework that offloads the control applications by separating the controllers into two different roles such as a root controller and a local controller. The root controller processes rare events and while highly replicated local controllers cope with frequent events. The local controllers are not connected each other. Therefore, it only handles the local events that require the local visibility. However, since the root controller maintains the network-wide global visibility, the root controller involves in packet processing that requires the global network state. Park et al. [70] proposed a novel solution, called RAON, that recursively abstracts controller's underlying networks as OpenFlow switches to reduce the complexity. In this architecture, the networks of the lower-level controllers are abstracted as a big OpenFlow switches. This abstraction extracts the relationship between two different networks that are operated by physically different controllers. Therefore, all the ingress and egress ports of the network becomes the ports of logical OpenFlow switch. Lee et al. [53] proposed a hierarchical controller structure with a super controller that collects global visibility from the lower-level controllers. Their main contribution is defining northbound message formats to realize the hierarchical controller in the field. They defined three different types of messages; normal messages, bandwidth event messages, and delay event messages.

New types of messages such as bandwidth_event messages and delay_event messages are added in order for a super controller to quickly respond to abnormal events from the underlying network operated by the lower-level controllers.

CHAPTER 3

MEASUREMENT AND ANALYSIS OF AN ACCESS NETWORK AVAILABILITY

Before we dive into the SDN's practical management issues, which include high availability issues, we discuss the network availability of a traditional network. In this chapter, we present our work on the measurement and analysis of the access network health. Understanding the health of a network via failure and outage analysis is important to assess the availability of a network, identify problem areas for network availability improvement, and model the exact network behavior. However, there has been little failure measurement and analysis work on access networks. We carry out an in-depth outage and failure analysis of a university campus network (University of Missouri-Kansas City) using a rich set of node outage and link failure data and topology information. We investigate the attributes of hardware/software and misconfiguration problems of the networks, the relation of link failure and node outage, the node availability, and the correlations between layers of a hierarchical network. For this dissertation, we mainly focus on the campus network availability. Section 3.1 describes the architecture of the campus network and the data sets we used for the availability measurement and analysis.

3.1 Campus Network Architecture and Data Sets

The campus network of our study is designed in a hierarchical manner that is a common practice of campus or enterprise networks [10]. It provides a modular topology of building blocks that allow the network to evolve easily. A hierarchical design avoids the need for a fully-meshed network in which all network nodes are interconnected. The building block components are the access layer, the distribution layer, and the core (backbone) layer as shown in Figure 10. The building blocks of modular networks are easy to replicate, redesign, and expand. There is no need to redesign the whole network each time a module is added or removed. Distinct building blocks can be put in-service and taken out-of-service with little impact on the rest of the network. This capability facilitates troubleshooting, problem isolation, and network management. In a hierarchical design, the capacity, features, and functionality of a specific device are optimized for its position in the network and the role that it plays. The number of flows and their associated bandwidth requirements increase as they traverse points of aggregation and move up the hierarchy from access to distribution and to core layer.

In the earlier years - until 2007, the UMKC network had 2 core routers in the core layer, 38 routers in distribution layer, and 373 nodes in the access layer. Since then, the core layer has increased up to 3 routers with different names. The new core router was added more recently to aggregate some part (e.g., dormitory area) of our campus network and wireless. In the distribution layer, there are currently 54 routers. The access layer has about 571 nodes and includes wireless access points, switches that connect to end-systems directly, and switches that aggregate other switches.

We have collected the node outage data as well as the link failure data from the university campus access network. As for network topology, we have had the direct and complete network topology information available for the network operators. We have

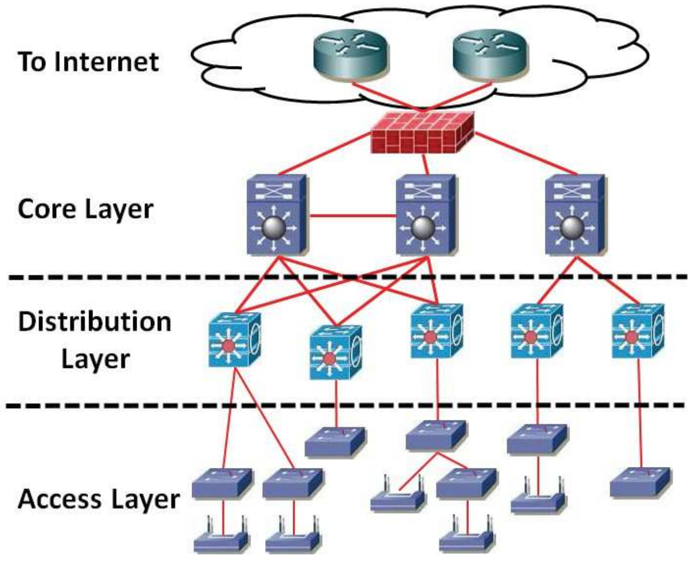


Figure 10: Hierarchical access (university campus) network design

used the naming conventions of devices to classify and relate devices, and utilized the topology information tool, called 'Intermapper'. Additionally, we have incorporated vendors' documents in regards to the causes and recommended actions, and have discussed the network operators' anecdotal comments on special events and actions. To the best of our knowledge, those data are the most extensive and complete data used in network failure and outage analysis.

Node outage data was gathered by SNMP polling and trap, and it is from April 7, 2005 till April 10, 2009 with 42,306 outage events. The polling time varies depending on the monitored devices ranging from 2 minutes to 5 minutes. The outage event time is recorded in the unit of minutes, and the outage duration is measured with second granularity.

Link failure data is collected from syslog which is UPDOWN messages from each device to a central syslog server. The period of data is from October 1, 2008 to October 5, 2009. Among many syslog error messages, we only consider 'LINK-3-UPDOWN' messages as pertaining to failure analysis. The 12 month data contains roughly 46 million syslog messages, of which 3.8 million messages represent 'LINK-3-UPDOWN' attached to itself. Syslog data has a slightly different format depending on the device vendors and router OSes. The campus consists of routers and switches from mostly Cisco, providing a similar format of syslog messages. Note that a link failure can occur due to software/hardware malfunction, natural or human-caused incidents, and may not lead to service outage due to redundancy or recoverymechanisms.

There may be some possible artifacts in the data, however, due to in-band (the monitoring data follows the same physical path as the user data) monitoring, the SNMP polling interval, and nature of protocol. Failure or outage reporting can be affected by the topology of the network. Any failure that is on the path to the monitoring system would likely result in an outage being reported for all devices on the path, though it is possible that the issue only affected one host. If connectivity is lost between the sending device and the syslog server, the syslog event would not be recorded. Additionally, as syslog uses UDP protocol, data can possibly be lost due to transient network congestion, CPU load, OS patching, EIGRP reconvergence, STP (Spanning Tree Protocol) recalculation, etc.

3.2 Network Availability Measurement

In this section, we investigate the availability of network nodes over four years. The node availability is the percentage of the uptime of a node. For each node i, let $NodeAvail(i)$ represent the node availability over a month, and it is computed as below.

$$NodeAvail(i) = \frac{TTBF(i) - TTTR(i)}{TTBF(i)} \times 100 \tag{3.1}$$

where $TTBF(i)$ is the monthly Total Time Between Failure of node i, and $TTTR(i)$ is the monthly Total Time To Repair of node i. This formula shows that we can improve the network availability by increasing the time between failures and reducing the time to recover. As we will see from Chapter 4, we focus on reducing the time to recover. Then, we compute the mean node availability (MNA) of all the nodes in the network.

$$MNA = \frac{\sum_{i=1}^{m} NodeAvail(i)}{m} \tag{3.2}$$

where m is the number of nodes in the network. The data set of the TTR per device is derived from the node outage data shown in Figure 11 and the monthly mean node avail- ability for the period from April 2005 to March 2009 is shown in Figure 12. We only use the events of the unplanned outages. We exclude the planned outages from the results to see the impact of the unplanned outages on the network availability. We observe that the network maintains two or three-nine availabilities in most of the months. It appears to be fairly healthy performance, even though there is no comparable measurement, to the best of our knowledge. Delve into the details, we notice one big drop in the availability in August 2006. Consulting the network operator, we confirm that there was a fire near a building that took out the fiber that month. We also observe slightly lower availabilities in

Table 8: Long term outages in the access layer

Year	Month	Causes of Degradation
2006	Jun	Reason 1&2&3
	Aug	Fire accident
2007	Apr	Reason 2
	Dec	Reason 2
2008	Jun	Power outages all over campus
	Jul	Backup link installation & OS bugs
	Nov	Reason 2&3

several months in 2007 and 2008. To concisely explain those occasions, we list possible reasons for the unidentified outage events below, based on the consultations with the net- work operator. Then, we summarizes the causes that made the performance degradation for each case, in Table 8.

- Reason 1: Issues that were either out of our control to correct any more quickly (like power problems)

- Reason 2: Issues that didn't justify an on-call response, thus were dealt with in the morning
- Reason 3: Issues that we were working on but took a while to fix
- Reason 4: Issues that affected monitoring but not operation

3.3 Network Availability Analysis with Splunk

We also discuss network availability analysis using Splunk and tailored scripting. Splunk is one of big data analysis tools and provides easy classifications and statistics

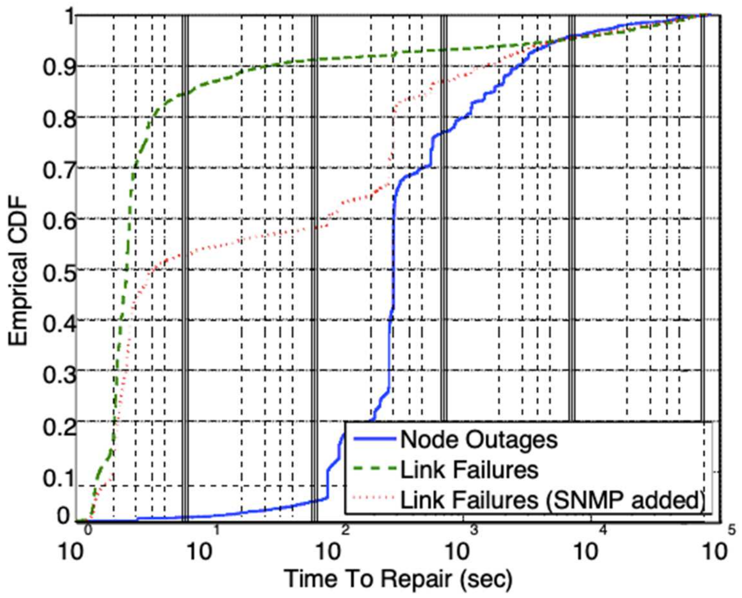


Figure 11: Node Outages vs Link Failures.

in a convenient format by efficiently capturing, indexing, and correlating big data. It analyzes the similarity between each line of the given data and recognizes the format of the messages or anomalies. It is very useful to quickly check various statistics of big data in real-time. Therefore, it enables us to have an agile visibility on the data and manage systems efficiently.

As the size of the network increases, network operators usually focus on only important links that are uplinks from a switch to other switches in the upper layer. Considering the limited human resources, it's impossible for them to track all the network messages caused by the very end links due to the sheer amount of messages being generated daily. Currently, the issues with individual interfaces are not monitored well nor

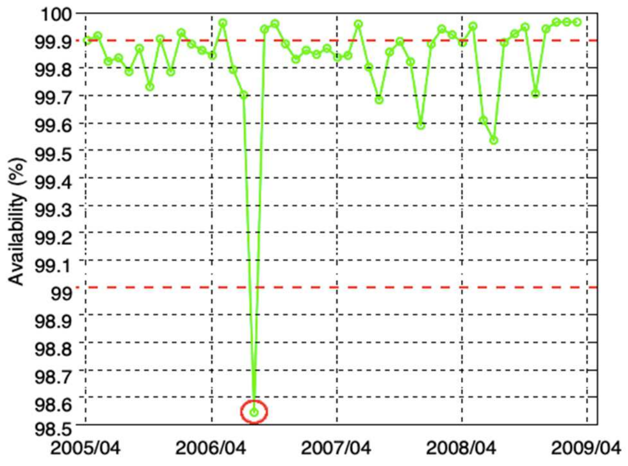


Figure 12: Node Availability (SNMP).

fixed unless a user contacts the network operators. However, to improve the user experi- ence, we need to harness the syslog messages by providing an automatic tool that analyzes network log messages and detects detrimental network events based on the institutional network policies.

In order to quickly identify network anomaly, we conduct quantitative analysis that ranks the number of node outages and link failures. We use Splunk for this analysis so that we can identify the problematic areas in our campus network taking the spatial and temporal aspects into consideration. For example, as shown in Figure 13, Splunk identifies that our network has many node outages in the "*D*" field of our campus. This is a soccer field, which is a wide-open area. Since no students expect Wi-Fi availability in this area, no complaints have been filed and it was left unfixed. Splunk can also be used to detect a problematic network component. In Figure 14, Splunk indicates that we have many link failures in one of switches in the "*m*" building. The possible reasons could be related to a bad port on the switch, a bad adapter on a client's NIC, or very old cables such as CAT3. In this case, the old cables cause these errors. After the new wiring installation, these problems are resolved. This type of errors has detrimental impact on only individual network devices, which is why it doesn't get urgent attention.

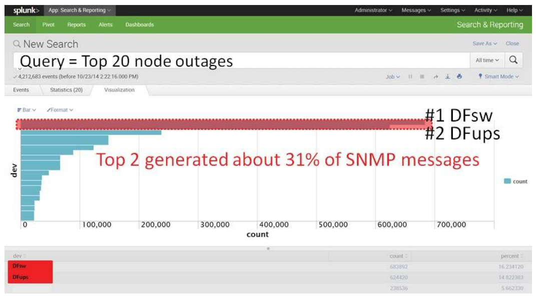


Figure 13: Statistical analysis of syslog in Splunk

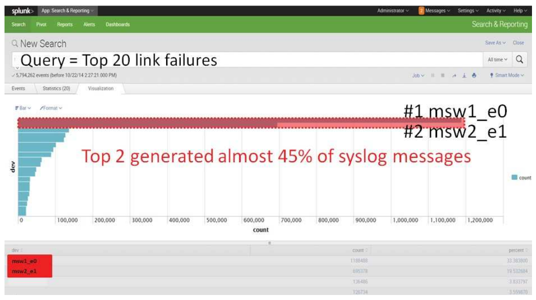


Figure 14: Statistical analysis of syslog in Splunk

These network events that are captured by Splunk are hard to be monitored by network operators since these errors don't have significant impact on the network operation. There is no way for them to get this information unless they carefully look through all the node outage and link failure events. Network availability analysis with Splunk can help network operators actively search problematic areas and devices in a quick and easy way.

3.4 Summary

We have conducted analysis of node outage and link failure data on a university campus network (UMKC) in order to understand the general characteristics of an access network including network availability. In order to precisely analyze the characterizationof the campus network, we incorporated vendors' documents in regards to the causes and recommended actions, and the network operators' input on special events and actions as well as long periods of network data such as syslog messages and SNMP data. This study on a campus network provides insights on the behaviors and conditions of access network availability, and potential end-to-end availability expectations.

CHAPTER 4

SDN CONTROL PATH NETWORK RELIABILITY

In this chapter, we present our approaches to the current high availability problems. We elaborate our proposed *control path* reliability management framework that includes several algorithms and describe its implementation. As aforementioned, the remote and centralized controller needs to be connected to its underlying network devices and communicate with them to manage flow requests from the network and impose network administrative policies into the network. This introduces new physical links between the controller and network devices. In addition to this, the controller can be configured as   a cluster having multiple controllers for high availability (as well as scalability). In this case, there would be a separate network that connects the multiple controllers in the clus- ter. In this work, we call the links that connect the controllers in the cluster as well as between the controller and underlying network devices *control paths*.

We propose a novel Multitemporal Cross-Stratum (MuCroS) discovery protocol framework, which efficiently orchestrates different reliability monitoring mechanisms over SDN networks and synchronizes the control messages among various applications. MuCroS facilitates multiple discovery frequency timers for each target over different stratum instead of using a uniform discovery timer for the entire network. It supports many common reliability monitoring factors for registered applications by analyzing offline and online network architecture information such as network topologies, traffic flows, virtualization architectures, and protocols. MuCroS framework consists of a high availability registration platform (HARP) and the topology-aware reliability (TAR) and traffic-aware discovery (TAD) protocol facilities. MuCroS framework is implemented on both Ryu and Cisco's OpenDayLight (ODL) controllers. Extensive Mininet experimental results validate that MuCroS significantly improves discovery message efficiency and makes the control traffic less bursty than OFDP with a uniform timer. It also reduces the network status discovery delay without increasing the control overhead.

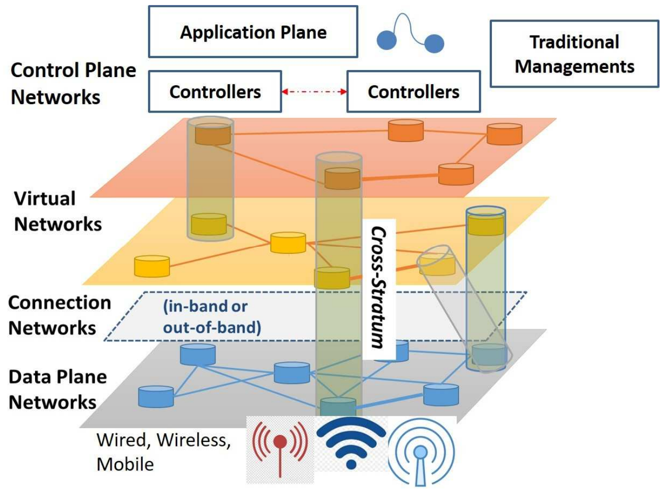


Figure 15: Complex SDN Reliability Management Domain

4.1 Network Reliability Issues

Network reliability management is one of the most crucial operational functions for the network service providers (NSPs). A network system fundamentally uses various heartbeat based failure detection mechanisms which is built-in distributed network devices. No heartbeat from a remote or neighbor node over a threshold duration indicates a potential failure of the node or link. The recent softwareization of network functions, controls, and applications is promising, as it improves the cost efficiency, control accuracy, and deployment flexibility of infrastructures. Software-Defined Networking (SDN) is a softwareization technology that logically centralizes the application and control planes (controllers) of a network by separating them from the underlying data plane (forwarders). OpenFlow [62] has been adopted as a southbound communication protocol between the control plane and data plane networks. In SDN, where the control plane responsibility is moved to the logically centralized controller, the network reliability schemes are operated by the controllers as centralized discovery protocols. An SDN controller operates periodic heartbeats to discover the initial network topology. The controller maintains up-to-date network visibility of the discovered network topology using the remote node's status notifications and periodic discovery messages. For example, an SDN controller identifies SDN switches when they initiate a TCP connection to the controller according to the controller's configuration. In addition, it discovers the network links by using discovery protocols such as the Link Layer Discovery Protocol (LLDP) [63], Broadcast Domain Discovery Protocol (BDDP) [64], and OpenFlow Discovery Protocol (OFDP) [65]. The SDN controller also finds connected hosts when they initiate a new flow message (i.e., initially ARP and RARP). Among them, LLDP is used as a dominant periodic discovery protocol.

However, the issue of centralized reliability management in softwareization architecture becomes more complex due to multi-lateral new network domains and poses both scalability and latency challenges on the existing network reliability mechanisms in order to achieve the same reliability services. The fundamental failure sensing timing and sequence can be vary and unnecessary redundant messages are often induced depending upon the given network architecture such as net- work topologies, virtualization, protocols, flows, policies, etc. In some situation, the control messages are conflicting each other that causes consolidation or convergence time. It not only incurs control overhead but also delays the topology and failure discovery processes. For example, in traditional networks, LLDP was configured as an optional protocol for the link layer neighbor discovery. However, SDN relies heavily on LLDP for discovering and maintaining its network visibility. Each controller (i.e., Cisco's ODL) maintains a uniform period timer for a discovery protocol. The total number of LLDP messages for a controller to process for an OFDP discovery period is about twice of the entire number of switch ports (including the inter-SDN switch port, host port, and non- SDN switch port) in the network. Hence, the control message scalability decreases significantly, if the network size and the discovery frequency increase. As illustrated in Figure 15, if the forwarding network is an in-band or tree topology and is used by various virtualized networks, many redundant control messages will be introduced to the network especially for the top of the network switches. Although there have been a few recent SDN reliability studies in the forwarding links [66] and control cluster [67], [68], little work has been conducted to orchestrate the separated abstraction and virtualization network planes in support of the SDN reliability.

In this paper, we propose a Network Architecture-aware Reliability Management Schemes (NetAware) to efficiently orchestrate different reliability monitoring mechanisms over multi-lateral SDN network architecture and synchronize the control messages among different controllers and applications. NetAware uses the network architecture information to expedite the network reliability decision as well as reduce any unnecessary redundant control messages. NetAware facilitates many common reliability monitoring factors for the registered applications by analyzing both off-line and on-line network architecture information such as network topologies, virtualization, protocols, flows, policies, etc. NetAware consists of a High Availability Registration Platform (HARP) and a Topology Aware Reliability (TAR) discovery facility. HARP handles redundant management messages due to the network virtualization and aligns asynchronous reliability protocols. TAR promotes hierarchical network topology in an order of importance based on the core, aggregate or edge. It differentiates the discovery message period according to the network topology. The impact of a target (node/link) to the network traffic is calculated according to the location, relationship, and functionality (i.e., core, aggregate and edge). For example, in a data center environment, a typical network architecture uses a three-tiered design that has a core tier in the root of the tree, an aggregation tier in the middle, and an edge tier at the leaves of the tree (i.e., Top of Rack) [69]. TAR can also facilitate many common reliability monitoring parameters such as protocol type, heartbeat mechanism, period, and target for the registered applications by analyzing both off-line and on-line network topology information. We implemented both HARP and TAR into Cisco's Open-DayLight (ODL) [71] High Availability (HA) component. By taking a common corrective action against a failure, it acts as an effective decision-making tool. The Mininet based experiment results show that NetAware eliminates redundant control messages over different virtualized network management segments, uses much less periodic control messages, and expedite a failure detection on the critical network segment without impacting the network scalability.

4.2 Motivations

4.2.1 SDN Control Message Experimental Setting

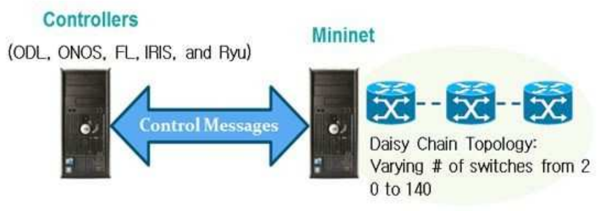


Figure 16 Mininet Configuration

As illustrated in Figure 16, we have conducted initial control message experiments using Mininet [15] with daisy chain based networks by varying the switches between 20 and 140 as well as tree topology networks varying the tree depth from 2 to 5 levels (assuming 2 fan outs, there are 3, 7, 15, and 31 switches, respectively). We have captured control messages for 10 mins with various SDN controllers including ODL, ONOS [72], [73], Floodlight [74] (FL), RYU, IRIS.

In this observation, we can infer a few interesting SDN controller design approaches. The results indicate that each SDN controllers interpret the same OpenFlow specification differently. Some controllers use far more initial control mes- sages. Also, it should be noticed that there are many redundant control messages such as echo, LLDP, and ICMP to monitor network status for detecting and isolating network failures as well as to identify the network topology. They are not synchronized among the applications. They often cause more complex detection and isolation process and potential racing conditions among the network status.

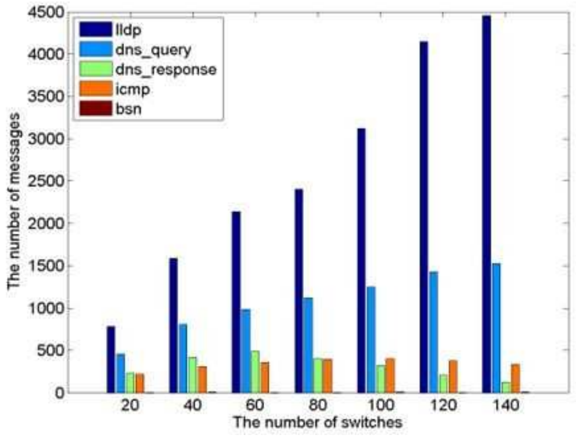


Figure 17 Total Control Message

According to Figure 17, there are various initial control messages to setting up the network. However, among the control messages, periodic LLDP messages are dominant. Figure 18 shows that ODL generates the smallest number of control messages. ONOS has a comparatively small number of control messages, which increases linearly as the number of switches increase. On the other hand, RYU has a relatively large number of control messages and the number of control messages increase remarkably as the number of switches increase. This suggests that an RYU controller can be easily congested with its own control messages.

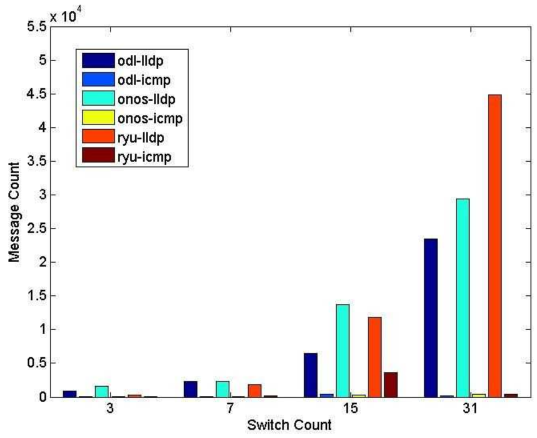


Figure 18 Control Messages Per Controller

4.2.2 Observation on Asynchronous and Redundant Reliability Management

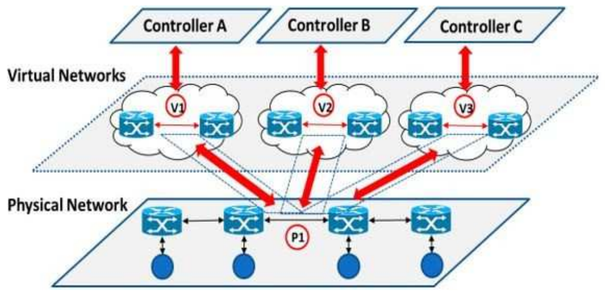


Figure 19 Redundant LLDP Messages over Virtualization Layer

As one of the essential problems in the network virtualization, virtual network embedding (VNE) has been widely studied to achieve different goals. Various VNE models have been proposed with different optimization goals or constraints. However, as layers of virtualization, policy application, and service chaining rely on network visibility, virtualized network management becomes more critical issues and impacts more complex ways. There are several management strategies to maintain the network visibility such as Event Driven (Report every events, the Health Manager (HM) will take care of them), Polling (Reply the event only if an HM asks), and Callback (Report the registered events only). However, it is unclear which approach works better in which condition. As shown in figure 17 there also are many periodic control messages such as echo, LLDP, RARP, and ping to monitor network status in order to detect and isolate network failures as well as to identify the network topology. However, many messages redundant in their functionality and are not synchro- nized among the applications. They often cause more complex detection and isolation process and potential racing conditions among the network status. For example, as illustrated in Figure 19, LLDP messages for the virtual networks V1, V2, and V3 will be used for each virtual network management. However, all of those LLDP messages are used redundantly, indeed to discovery the link status of P1 in the physical network. In addition, as shown in Figure 20, multiple control messages (LLDP, ICMP, RARP, etc.) are asynchronously used for the same component for the same status detection. For an efficient reliability management, a facility that can be aware of the virtualized network mapping in order to optimize the control over the heterogeneous softwareized networks. The framework should be able to synchronize control messages via registration methods as well as orchestrate control messages over the network virtualization components. For example, if an SDN network segment is an in-band network and SDN controller is aware of the network topology, it can send a representative discovery message instead of sending discovery messages for the overlapping network segments. Only if the representative discovery fails, it can conduct an individual discovery process.

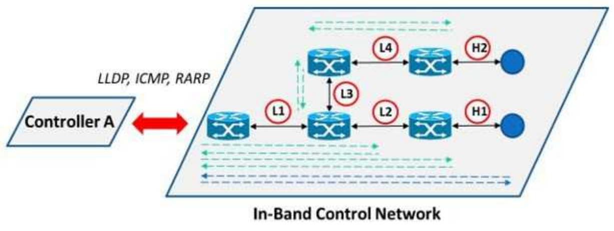


Figure 20 Redundant Reliability Protocols over In-band SDN Network

4.2.3 Issues on Topology Agnostic Discovery Protocols

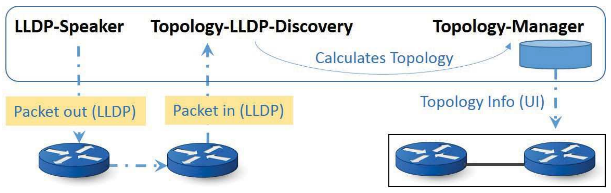


Figure 21 Topology of LLDP-Discovery and LLDP-Speaker

The LLDP has been used by distributed network devices for advertising their local information such as identity, capabilities, and neighbors. In traditional networks, this vendor-neutral link layer protocol is configured as an optional component in network management and monitoring applications. However, in SDN, OFDP, or an SDN LLDP is a centralized discovery protocol that transmits information about the current status of a device and the capabilities of its interfaces. As illustrated in Figure 21, the SDN controller has LLDP facilities including an LLDP Speaker and LLDP Discovery. When a switch is connected to a controller, an LLDP Speaker periodically sends dedicated LLDP packets in the Packet-Out messages for all the interfaces of the network switches. The switch floods the LLDP packets through all of its ports. Upon receiving an LLDP packet, an SDN switch sends a Packet-In message to its controller for acknowledging a direct link between the switches. Figure 22 presents accumulative LLDP message counts on the daisy chain topology networks with 2 and 8 switches, respectively. As the network size increases, the amount of control messages dramatically increases due to the path dependency among the network switches. For example, if an SDN network segment is an in-band network or over the tree topology, a link failure on L1 can have a more significant impact on the entire network than a link failure on L2 because more components are depending upon the L1 link. Hence, increasing the LLDP discovery frequency will worsen the scalability of the network, but decreasing the LLDP discovery frequency will delay the detection of the network status changes.

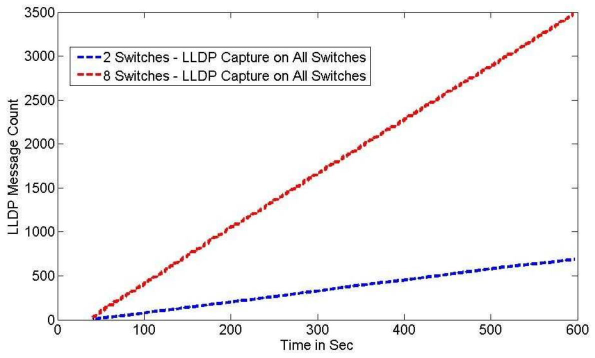
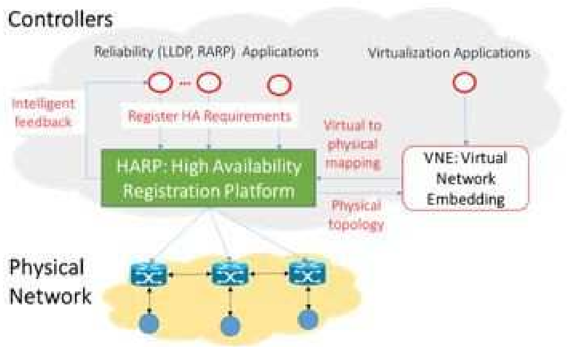


Figure 22 LLDP messages over the daisy chain network

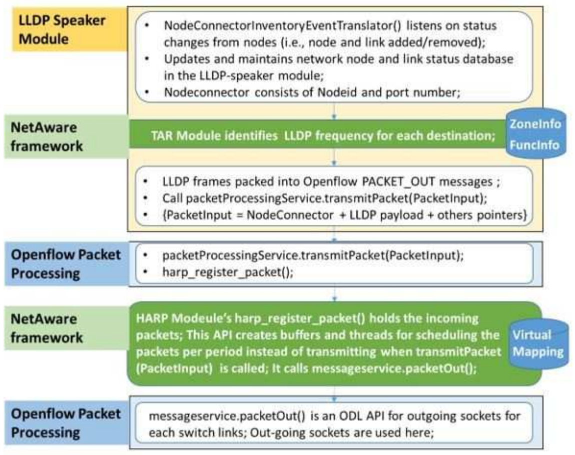
4.3 Reliability Management Framework



Controllers
Reliability (LLDP, RARP) Applications
Virtualization Applications
Intelligent feedback
Register HA Requirements
Virtual to physical mapping
HARP: High Availability Registration Platform
VNE: Virtual Network Embedding
Physical topology
Physical Network

Figure 23 HARP Architecture

We design and develop an intelligent SDN reliability control approach. The Network Architecture-Aware Reliability (NetAware) Management Framework facilitates many com- mon HA monitoring factors for the registered applications by analyzing both offline and online information instead of implementing HA services for each object (i.e., application, controller, protocol, and switch). NetAware eliminates redundant control messages, simplifies the fault detection and isolation process, and expedites fault recovery by chaining network problems intelligently. It enables a common corrective action against a failure. NetAware framework consists of TAR and HARP modules. We prototype NetAware into Cisco's ODL in support of a few practical case scenarios. As described in Figure 23, a High Availability Registration Platform (HARP) orchestrates different monitoring mechanisms over the com- plex SDN architecture and synchronize the control messages among different controllers and applications. In this case scenario, we implement HARP that filters redundant discovery LLDP messages over the virtual networks by aligning with the Virtual Network Embedding (VNE) information. HARP gets virtual to physical network mapping information from the VNE. Along with the current physical network status, HARP arranges the physical network control budget for the registered HA requirements. Virtualization application also takes physical network topology from the HARP for the next VNE. A Topology-Aware Reliability (TAR) module enhances the scalability and latency issues of the centralized discovery mechanism by assigning target specific discovery frequencies instead of using a uniform period for the entire network. It promotes hierarchical network topologies in an order of im- portance based on the core, aggregation, or edge. The impact of a target (node/link) to the network traffic is calculated according to the location, relationship, and functionality (i.e., core, aggregation, and edge). We used a zone concept to differentiate the discovery message period according to the network topology. We set a 3-depth binary tree topology and categorized the topology by zone. The node on the top of the tree topology, such as the core switch, is categorized as Zone1. The middle tier of the topology in an aggregation tier is categorized as Zone2. The edge nodes in the bottom of the tree topology categorized as Zone3. For example, a failure in a core switch may impact the most data traffic and control messages. By sending more frequent discovery messages to the core (the more important nodes), TAR can expedite a failure detection and recovery on the critical network segment while sending less frequent discovery messages to the edge nodes.



- NodeConnectorInventoryEventTranslator() listens on status changes from nodes (i.e., node and link added/removed);
- Updates and maintains network node and link status database in the LLDP-speaker module;
- Nodeconnector consists of Nodeid and port number;

TAR Module identifies LLDP frequency for each destination;

- LLDP frames packed into Openflow PACKET_OUT messages ;
- Call packetProcessingService.transmitPacket(PacketInput);
- {PacketInput = NodeConnector + LLDP payload + others pointers}

- packetProcessingService.transmitPacket(PacketInput);
- harp_register_packet();

HARP Modeule's harp_register_packet() holds the incoming packets; This API creates buffers and threads for scheduling the packets per period instead of transmitting when transmitPacket (PacketInput) is called; It calls messageservice.packetOut();

messageservice.packetOut() is an ODL API for outgoing sockets for each switch links; Out-going sockets are used here;

Figure 24 Network Architecture aware Reliability Management Implementation

Considering the network distance to travel, TAR can achieve faster detection without worsening the network scalability issues. As illustrated in Figure 24, the NetAware is implemented in ODL on the LLDP-Speaker module (TAR) and the Open-Flow Packet processing module (HARP). LLDP-Speaker is an application in the Openflow module for sending LLDP frames. A NodeConenctorInventoryEventTranslator( ) thread listens on status change events such as nodes and link added or removed from the networks. It maintains all the node-connector information that consists of the node ID and port number. The LLDP-speaker module runs a thread that sends the LLDP frames packaged into Openflow PACKET OUT messages to all learned nodes for every 5 secs. We intercept this routine to embed the TAR module that returns an LLDP frequency value for each target switch (switchID). The LLDP frequency () determines how often the LLDP-Speaker sends a probe to a specific switch for requesting the LLDP message. The frequency value can be dynamically assigned according to the network condition (i.e., a switch functionality and frequency). With the returned LLDPfrequency(period) and switchID(target), the TAR module calls a packetPro- cessingService.transmitPacket (PacketInput) API. The Pack- etInput consists of the NodeConnector, LLDP payload, and other pointers. In an OpenFlow packet processing module, the packetProcessingService. transmitPacket(PacketInput) calls harp_register_packet(). HARP module holds the incom- ing packets by listening harp_registe_packet(). This API creates buffers and threads for scheduling the packets per period (identified by the TAR module) instead of just transmitting the packet via transmitPacket(PacketInput). In HARP, virtual net- work embedding information is used to filter out unnecessary discovery requests. Then, it calls messageservice.packetOut() that uses an out-going ODL sockets for sending the packets out to switches. Algorithm 1 presents an LLDP discovery frequency function of the TARMan moudle that returns an LLDPfrequency value for each target switch (switchID). The LLDP frequency determines how often the LLDP-Speaker sends a probe to a specific switch for requesting the LLDP message. First, it calls the getZones function to check the right zone of the target. It reads a switch functionality from the configuration (ZoneInfo) using the switch ID. Using the switch functionality (i.e., core, aggregation, and edge), it assigns the zone number. Second, using the zone number, it calls the readFrequency function. It reads an LLDP control message period from the configuration using the zone value. The frequency value can be dynamically assigned according to the network condition. With the returned LLDPfrequency (period) and switchID (target), the TARMan module calls a packetProcessingService.transmitPacket (PacketInput) API. The PacketInput consists of the NodeConnector, LLDP pay- load, and other pointers. In an Openflow packet processing module, the packetProcessingService.transmitPacket (Packet- Input) function calls the messageservice.packetOut () that is, an ODL API to send the packets out to switches.

```
Algorithm 1 LLDP Discovery Frequency
Input: switchID
Output: lldpfrequency
 1: for all SwitchID  queue do
 2:     zone = getZones(switchID)
 3:     lldpfrequency = readFrequency(zone)
 4:         read a LLDP control message period from the configuration using the
        zone value
 5:     return lldpfrequency
 6: end for
 7: function GETZONES(rwitshI D)
 8:         read a switch functionality from the configuration using the switch ID
 9:     if readSWfunc(switchID)  core then
10:         return 1
11:     else
12:         if readSWfunc(switchID)  agg then
13:             return 2
14:         else
15:             return 3
16:                 edge                                        //
17:         end if
18:     end if
19: end function
```

4.4 Evaluations

4.4.1 Experimental Setup

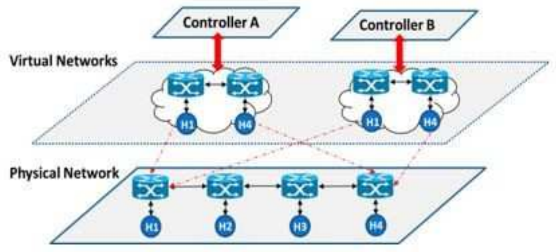


Figure 25 SDN Control Message Experiment Setting

We investigate our proposed architecture through emulation of real implementation. For the first emulation environment we used DELL PowerEdge T320 server with Intel (R) Xeon(R) CPU ES-2403 V2 @ 1.80 GHz x4 and Ubuntu 14.04 OS. OpenFlow controller, running on ODL Beryllium-SR2. On the second emulation environment we also used similar setup with DELL PowerEdge T320 server with Intel (R) Xeon(R) CPU ES-2403 V2 @ 1.80 GHz x4 and Ubuntu 14.04 OS. We had OpenFlow controller, running on TARMan. We have conducted five experiments: 1) testing LLDP message overhead for the virtual networks, 2) testing LLDP message overhead from the controller I/O, 3) testing accumulated control message hop counts by capturing LLDP messages from the switches, 4) testing the impact of control message outage over the in-band networks, and 5) testing the impact of data flow outage. For aforementioned experiments, as illustrated in Figure 11, we built a balanced binary tree topology network with a depth of three (7 switches and 8 hosts) by using Mininet. Ping is used to generate packets and Wireshark is used to capture messages from the loopback interface. The ODL controller uses the default 5 second uniform intervals in sending LLDP PACKET OUT messages to the switches, named "Without NetAware" or "ODL (5,5,5)". In the TAR module, we configured a couple of simplified frequency sets. The TAR (1,3,5) set is with intervals of 1, 3, and 5 seconds for the core, aggregation, and edge switches, respectively. The TAR (1,3,5) set creates the more frequent LLDP messages for the core and aggregation than the ODL (5,5,5) set in order to detect failures faster from those important links or nodes. The TAR (1,5,10) set is with intervals of 1, 5, and 10 seconds for the core, aggregation, and edge switches, respectively. The TAR (1,5,10) set creates the more frequent LLDP messages to the core switches than the ODL (5,5,5) set to have the quicker updates from the core links or switches. However, it creates the less frequent LLDP messages to the edge switches than the ODL (5,5,5) set that could have the slower updates from the edge links and switches. We assumed 3 consecutive LLDP message failures to change a status.

4.4.2 Control Messages Overhead for Virtual Networks

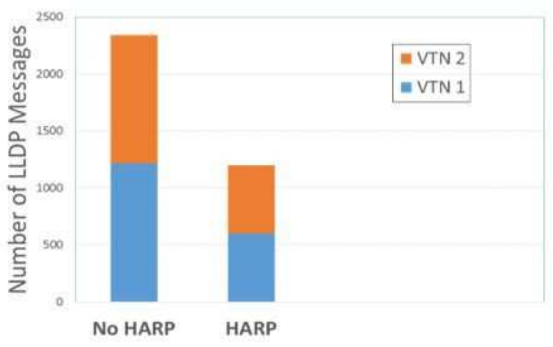


Figure 26 Control Message Overhead for Virtual Networks

We performed this experiment to appreciate the discovery message overhead over the virtual networks. As illustrated in Figure 26, we have created a daisy chain topology with four switches. Two OLD controllers are connected to Mininet. In each ODL controller, a virtual topology (VTN) is created. Both VTN connect hosts from 1 to 4. Wireshark capture is on the controller interface side. The number of LLDP messages for 2 VTNs with no HARP filter is two times greater than the number of LLDP messages with HARP VTN filter. In the HARP filter, we randomly (alternatively) filtered messages from each controller.

4.4.3 Control Message Overheads

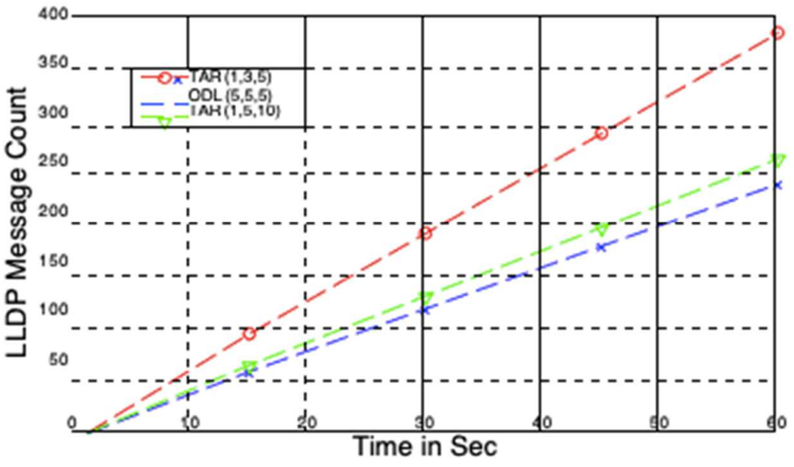


Figure 27 LLDP messages captured on a controller

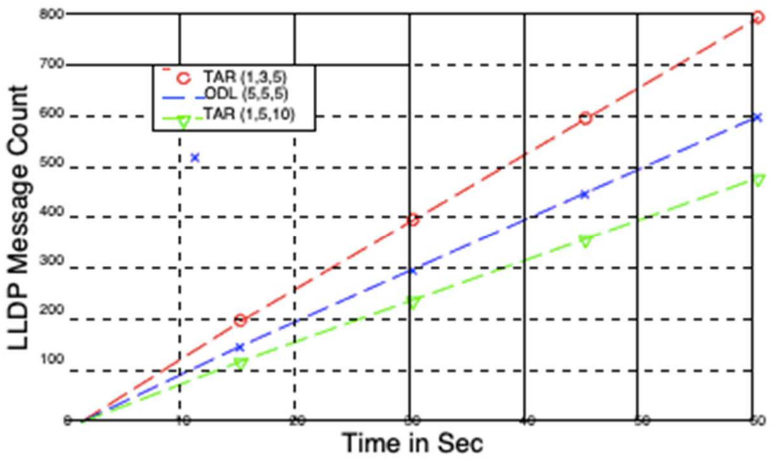


Figure 28 Accumulated LLDP messages captured by the switches

According to Figure 28, it is obvious that both the TAR (1,3,5) and TAR (1,5,10) sets create relatively more LLDP messages from the controller than the ODL (5,5,5) set. However, the LLDP messages are not significantly increased. Especially, the TAR (1,5,10) set generates the similar amount of LLDP messages to the ODL (5,5,5) set. On the other hand, Figure 15 presents the practical network usage (accumulated LLDP messages per network hop) by the LLDP messages in case of the in-band networks. The presented results are the accumulated LLDP messages captured by the network switches. Although the TAR (1,3,5) set still have slightly higher accumulated LLDP messages than the ODL (5,5,5) set, the TAR (1,5,10) set creates less accumulated LLDP messages than the ODL (5,5,5) set. This is because the important core and aggregation nodes are closer to the controller with the in-band network, which creates less accumulated LLDP messages. These results are promising because creating more frequent messages to the core does not create a significant control message overhead.

Figure 27 compares the performance of our configurable TARMan with the existing ODL in terms of the number of unique LLDP messages, in a balanced binary tree topology with 3 tier depths. We further measured the number of LLDP messages that are seen in all links in the network while traversing paths from/to the controller to/from switches in Figure 28.

For example, for ODL in 60 seconds there will be 12 LLDP PACKET_OUT message sent to the root and 12 LLDP PACKET_IN is received from the other side of the link where there is one hop from the controller. Therefore, ODL has total of 24 LLDP message multiplied by hop count (1) results to 24. The second depth in the binary tree requires 2 hops from the controller. There are 72 LLDP messages to those nodes from the controller in 60 second. The Total LLDP messages (72) multiplied by number of hops (2) gives us 144 messages. To the edge of the network there 144 total LLDP messages in 60 seconds multiplying that with the three hops required from the controller, we get 432. The sum of those three results in 600 LLDP messages count by hops for 60 second from the ODL controller. On the other hand, TARMan generates the most LLDP message count in the core of the network, where within 60 seconds there are 60 LLDP PACKET_OUT and 60 LLDP PACKET_IN messages reaching to total of 120 LLDP messages. Multiplying the 120 LLDP messages with hop count (1) results in 120 LLDP message for the core. At the aggregate there are 72 LLDP message times by the hop count (2) we get 144 LLDP message count. The edge of the network generates 72 LLDP message count. Multiplying the 72 with the hop count of 3 we reach to 216 LLDP messages. Summing the three, we achieve 480 LLDP messages for TARMan for the duration of 60 seconds. From this experiment we can conclude that, analyzing the LLDP message count between ODL and TARMan, we can observe that TARMan pays closer attention to the core, the node that is the closest to controller, by configuring the LLDP message more frequently. Even with more messages at the core, TARMan generates far less overall LLDP message count by hop than ODL. TARMan provides more realtime information of the core of the network resulting in more reliable controller as illustrated in Figure 28.

4.4.4 Impact Factor on Control Messages

It is important to identify failure quickly in order to begin the repair process as soon as a failure is discovered. We established connection of the software-defined-network to a controller. Our goal in this experiment is to measure the link or node failure impact on the control in-band and data flow. In an event of node failure or link failure, controller is impacted differently than the data flow. For example, when the core node is down, controller may lose access to the rest of the network. However, data flow between the active switches can still take place. To calculate the Impact factor for the controller, we used the number of impaired node multiplied by the outage time in second. Using our TARMan platform we configured the core, aggregate, and edge with two variations. One of the TARMan was configured with 1, 3, 5 seconds for core, aggregate, and edge respectively and the second was configured with 1, 5, 10 second intervals for core, aggregate, and edge respectively. We compared that with a controller without TARMan default of 5 second frequency for all core, aggregate and edge. We define an Impact to be the number of impaired/impacted node multiplied by the outage time in second.

For example, the TARMan platform that was configured with 1 second for the core failure would result impacting all 7 nodes. To calculate the Impact factor, we multiply 7 by outage time of 3 second (3 times the frequency of 1 second) results with 21. At the aggregate of the tree topology there will be 3 nodes impacted multiplied by outage of 9 second (3 times frequency of aggregate 3 second) would be 27. The edge the topology would have one node impacted multiplied by 15 second outage (3 times 5) would end up to be 15. Figure 29 illustrates the numerical values.

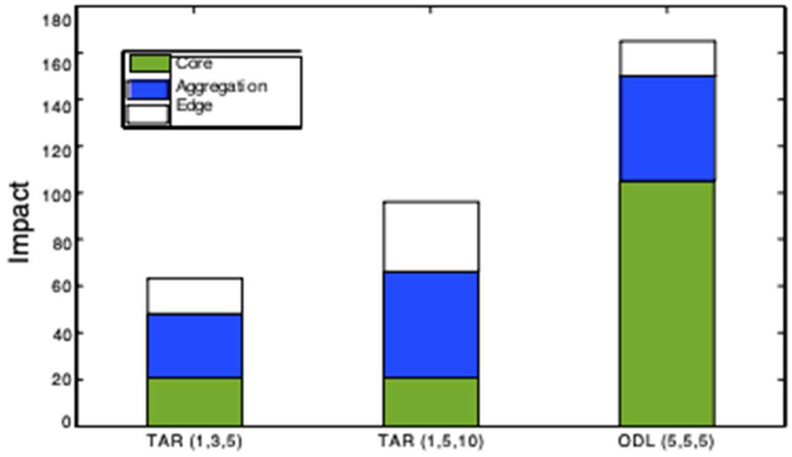


Figure 29 Impact on Control Messages

4.4.5 Impact Factor on Data Flow

In this experiment we calculate the Impact Factor on data flow. To calculate the impact factor, we utilized the number of impaired bi-directional flows multiplied by the outage time in seconds. To calculate the outage time, we use 3 times the frequency of the LLDP message. The controller assumes the node inactive after sending three consecutive LLDP message out and not receiving reply. We used softwarize network with TARMan and without TARMan. The TARMan platform was configured with two variations for core, aggregate and edge frequency. One was set with 1, 3, 5 seconds frequency for core, aggregate, edge respectively while the second variation of TARMan had 1, 5, 10 seconds for core, aggregate, and edge respectively. The controller without TARMan had the default frequency of 5 second. We used perfectly balanced tree topology with 3 depths. For our tree topology, there were total of 56 bi-directional flows form 8 hosts. To generate the bi-directional flows, we utilized Ping. Ping is a simple echo protocol. It works by sending a message to a server that contains the message "PING" and receive a copy of the message back. This allows the host that sent the ping to calculate the "Round Trip Time" (or RTT) that it takes a packet to reach a particular server. It calculates how long it takes for the packet to reach the server and to be sent back. It is also a method used for checking if a host is connected to a network. In our experiment, we utilized the ping protocol after simulating the node outage to measure the number of impaired flows. A core node failure would result in theoretical 28 bi- directional flows impacted by the outage. 28 multiplied by the 3 second outage (3 times 1 frequency) would be 84. The result of our experiment shows in section Bi-directional data flow result. We can observer that the Pings sent from h1, h2, h3, and h4 where only successful to each other and all other failed. This create two islands of networks in our tree topology and we can observe that 32 out of 56 flows were impaired causing 57% of the flows to drop. A core node failure in our tree topology creates two islands of network The two islands of network can communicate internally. However, they cannot communicate with each other. An aggregate node failure where theoretically impacts 14 bi-directional flows show different result in our experiment. We observed that 71% of the flows were impaired as a result of aggregate node failure. Theoretically, multiplying the 14 impaired flows by 9 sec of outage (3 time 3 seconds) would be 126. From our experiment, we see that 40 of the 56 flows were dropped. We can observe that while h1 can communicate with h2 and h3 can communicate with h4, they cannot reach the rest of the network or each other (h1 and h2 cannot communicate to h3 and h4). Theoretically 7 bi- directional flows will be impacted. 7 multiplied by 15 seconds (3 times 5 seconds) would be 105. The result of our experiment showed something different. It showed that 43% of the flows were impaired. Out of 56 flows, 24 were dropped. Figure 30 shows the result of our analytical computations.

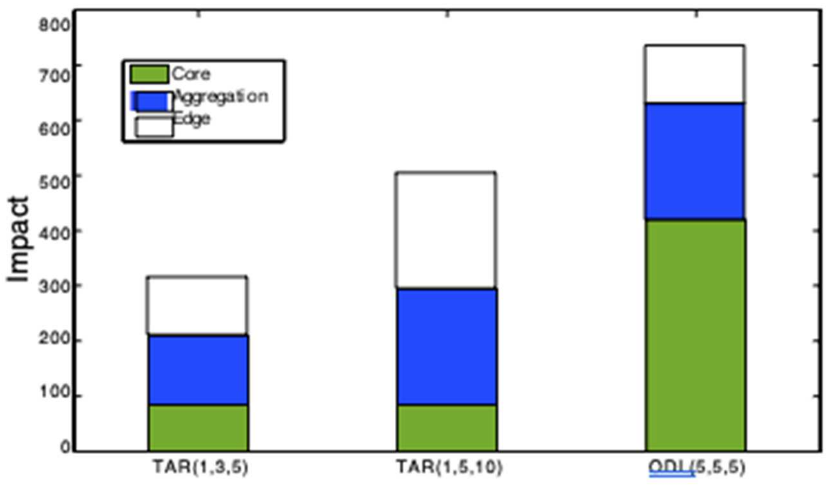


Figure 30 Impact on Data Flow

4.5 Conclusions

Little attention has been paid to the SDN network reliability management that suffers from scalability and latency issues. We proposed a novel Network Architecture-aware Reliability Management Schemes (NetAware) to efficiently orchestrate different reliability monitoring mechanisms over SDN network architecture and synchronize the control messages among different controllers and applications. We enabled the NetAware platform to provide fast and smart decision-making information for fast failure detection and recovery. A prototype is implemented on Cisco's OpenDayLight (ODL). Extensive experiment results exhibit that our algorithm achieves effective and efficient network failure detection while generating limited LLDP message overhead.

CHAPTER 5

SDN CONTROL PLANE NETWORK RELIABILITY

In this chapter, we discuss the reliability of control plane network. Despite tremendous software quality assurance efforts made by network vendors, chastising software bugs is a difficult problem especially, for the network systems in operation. Recent trends towards softwarization and open sourcing of net- work functions, protocols, controls, and applications tend to cause more software bug problems and pose many critical challenges to handle them. Although many traditional redundancy recovery mechanisms are adopted to the softwarized systems, software bugs cannot be resolved with them due to unexpected failure behavior. Furthermore, they are often bounded by common mode failure and common dependencies (CMFD). In this paper, we propose an online software bug detection, debugging, and isolation (BuDDI) middlebox architecture for software-defined net- work controllers. The BuDDI architecture consists of a shadow-controller based online debugging facility and a CMFD mitigation module in support of a seamless heterogeneous controller failover. Our proof-of-concept implementation of BuDDI is on the top of OpenVirtex by using Ryu and Pox controllers and verifies that the heterogeneous controller switchover does not cause any additional performance overhead.

5.1 Control Plane Network Reliability Issues

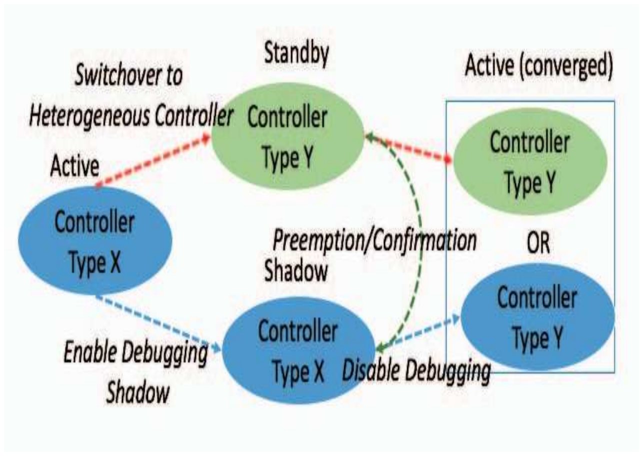


Figure 31 BuDDI N +2 Reliability

Software faults in the operating network systems can cause not only critical system failures [79] but also various unexpected and transient results. Although network vendors have a series of development guidelines, checkpointing facilities, assurance processes, and debugging mechanisms to improve their soft- ware reliability, it is commonly accepted that maintaining a bug free network system is impossible. The recent networking paradigm changes towards softwarization and virtualization has increased the detrimental effect of software faults on network functions, protocols, controls, and applications. Furthermore, the increasing open-source and third-party software (TPS) used in Software-Defined Networks (SDN) aggravates software bug problems because vendors may not fully identify all issues during software quality assurance testing. To cope with the software reliability issues, a few software bug handling mechanisms have been proposed. For example, the bug tolerant router design has been proposed in [79], [82] to run multiple diverse copies of virtual router instances. LegoSDN to tolerate SDN application failure [80], [82] focuses on SDN-App failures, fail-stop crashes and byzantine failures. However, software bugs are hard to be resolved with the traditional redundancy-based failure detection and recovery mechanisms alone as software bugs can cause unexpected root cause failures, baffle failure detections, and hinder recovery mechanisms. In addition, some of the deterministic bugs in SDN controllers are often bound by common mode failure and dependencies (CMFD) that bring the system into the same software failures after a failover. On-line debugging, and especially CMFD resolutions in SDN are still a relatively unexplored area. In this paper, we propose an online software Bug Detection, Debugging, and Isolation (BuDDI) middlebox architecture for SDN controllers. BuDDI consists of a shadow- controller based online debugging facility and a CMFD mitiga- tion module in support of a seamless heterogeneous controller failover. For on-line bug detection and debugging, unlike a traditional N +1 redundancy cluster system, we propose an N +2 load balancing cluster system where components (N) have at least two independent failover components (+2). As illustrated in Figure 31, BuDDI facilitates a CMFD mitigation module by taking advantage of software diversity of the existing heterogeneous controllers. In addition, BuDDI enables a shadow controller that mirrors the active controller functions and turns on a verbose debugging mode for a specific failure module. Eventually, the two failover components will converge into one active controller. If the shadow-controller cannot identify a software bug in a given period, it sends a preemption message to the active CMFD module to take over the active role. Otherwise, it will confirm an active role for the CMFD module.

Controller switchover algorithms and shadow controllers debugging facilities are built on the top of OpenVirtex, which provides the facility to create virtual networks and to map them to the physical network. The middlebox acts as proxy between the physical network and the controllers. As a preliminary part of our experiment, we choose two of the heterogeneous controllers (Ryu [76] and Pox [75]) to verify that both the heterogeneous controller switchover and N + 2 redundancy mechanism supports do not cause any additional performance overhead in the proposed BuDDI mechanism.

5.2 BUDDI

Our proposed architecture aims at bug detection, debugging and isolation (BuDDI) for SDN controllers. BuDDI is built on top of OpenVirteX, which is a network virtualization platform that enables operators to create and manage vSDNs [78]. BuDDI provides three main functionalities: (i) detection of any bug in the controller (ii) automatic debugging of the controller for bugs (iii) isolation by switching over the controllers. It works with all four main failure scenarios in the SDN deployment [80]: (i) controller server failure (hardware failure); (ii) controller crashes (bug in the controller code); (iii) network device failures (switch, application server or link failure); and (iv) SDN application (SDN-App) crashes (bugs in the application code). Our work focuses on the controller failures as well as on SDN-App failures. BuDDI is designed in such a way that whenever there is any failure in the Controller or the SDN-App crashes, it detects that the connection to the controller is down, and it switches over the controller. The same time debugging is also started. Important keywords are explained in Table 9.

Definition	Explanation
Active Controller	Controller to which the network is connected
Standby Controller	Controller to which switchover takes place after bug is detected in the active controller
Shadow Controller	Controller used for debugging. Copy of the active controller
Heterogeneous Controllers	Different types of controllers. E.g.,Pox + Ryu
Homogeneous Controllers	Same types of controllers. E.g., Pox + Pox.
Switchover time	Time between bug detection and connection to standby controller

Table 9 Switchover Mode Definition

A. Proposed Architecture

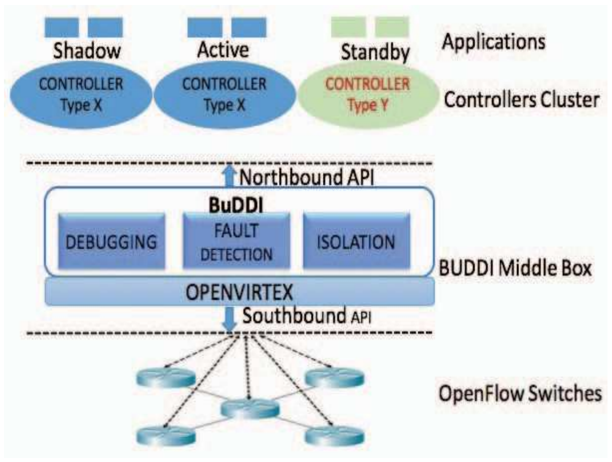
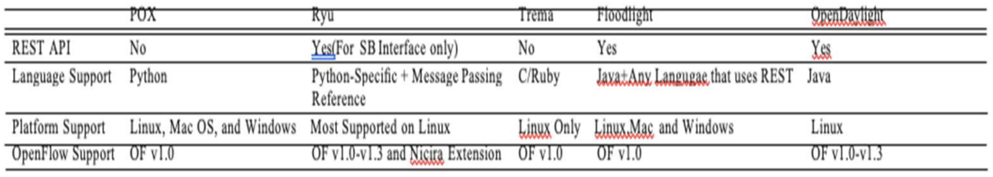


Figure 32 BuDDI Middlebox Architecture.

The proposed architecture is shown in Figure 32. The middle- box is connected to the controllers via northbound OpenFlow, whereas with the physical network, via southbound OpenFlow. Ryu, POX, FloodLight, Trema, and OpenDaylight are some of the most commonly used open-source controllers [9]. These controllers vary from each other in one way or another, which gives them diversity and supports our claim of using a heterogeneous controller approach. Table 10 lists the basic differences between the controllers.



	POX	Ryu	Trema	Floodlight	OpenDaylight
REST API	No	Yes(For SB Interface only)	No	Yes	Yes
Language Support	Python	Python-Specific + Message Passing Reference	C/Ruby	Java+Any Langugae that uses REST	Java
Platform Support	Linux, Mac OS, and Windows	Most Supported on Linux	Linux Only	Linux,Mac and Windows	Linux
OpenFlow Support	OF v1.0	OF v1.0-v1.3 and Nicira Extension	OF v1.0	OF v1.0	OF v1.0-v1.3

Table 10 Comparison among controllers

For this architecture there are three controllers used: active, standby and shadow controllers. The standby controller is a heterogeneous controller, whereas the shadow controller is a homogeneous controller. Failure in the controller can be caused due to indeterministic bug in the source code or due to a controller crash. When the active controller is down due to any bug in the SDN app, or due to any hardware failure in the controller, BuDDI switches the control from the active controller to the standby controller, which is a heterogeneous controller. We use heterogeneous controller since different controllers are coded in different languages and there is less probability that the same bug would reoccur. At the same time, the shadow controller, which is homogeneous controller, is also switched over, but is used for debugging the failure. The main aim of the shadow controller is to find the root cause of the failure. The shadow controller has the information about the application state, so if the automated debugging process passes the state at which the failure occurred in the active controller, then the shadow controller becomes the new active controller, or else, the standby controller continues to serve as the new active controller.

B. Functional Modules

Our architecture can be divided into three different functional modules as shown in Figure 32: (i) Fault Detection Module (ii) Isolation Module and the (iii) Debugging Module. Each module described below.

Fault Detection Module: The main function of this module is to monitor the applications running on the controller. Using the keep-alive messages exchanged between the data-paths and controllers, we try to monitor the connection between the virtual network and the controllers. If any online software bug causes the application to stop, an error message would be generated. By using this error message and the event log on the controllers, we can check which application on the controller has stopped or has encountered a bug. If there are any hardware failures or any network device failure, the keep- alive messages will not be exchanged and an error message would be generated.

Isolation Module: This module is responsible for switching over to a heterogeneous controller and the homogeneous controller at the same time. The heterogeneous controller will take over as the active controller whereas the homogeneous controller will be used for debugging purposes. The main reason for a switchover to a heterogeneous controller (in spite of applications being controller specific) is that if any non-deterministic software bug is found, it would be affected if we switched over to same type of controller. Since different con- trollers are coded in different coding languages, the probability that we will encounter the same bug in different controllers is less. This module is also responsible for the synchronization of the flow states.

Debugging Module: As we know, applications are controller specific. It will not be good to use a different controller for long time. So, we use the same type of controller as a shadow controller, which would be used in a debugging mode. An occurrence of a bug in an SDN application will most likely result in the SDN system being down. In order to seamlessly switchover and avoid failure, we use the shadow controller. The main idea is to see if we are getting the same error again in the shadow controller. If we encounter the same error in the same point, it would mean that there is a bug and it would be reported. If we do not encounter the same error in the shadow controller, it would mean that our active controller had some other failure and not a bug. The debugging module aims at detecting liveness bugs.

C. Switchover Procedure

Prior to the network getting connected to the controller, we need to have a copy of the active controller that would act as our shadow controller and also as a standby heterogeneous controller. Once the network is operational, it points towards the middlebox. Inside the middlebox, a virtual topology is created. The virtual topology points towards the active controller and the error detection module initialized to monitor the active controller. When any bug arises in the active controller, an error is generated in the error detection module. Once the error shows up in the error detection module, a notification is sent to the Isolation and Debugging module. Now the Isolation module will switch over the control of the entire network to the standby controller and transfer the flow states from the active controller to the standby controller. At the same time, it also makes a switchover to the shadow controller, which is an exact copy of the active controller, but this shadow controller would be used by the Debugging module to find the root cause of the bug. If while debugging the shadow controller, it does not find the same bug, which occurred in the active controller, it would be mean that there was some other error that occurred in the active controller, which was not necessarily a bug. Then the whole network would be switched over to the shadow controller and it would act as the active controller. If while debugging the shadow controller, the debugging module finds the same error that occurred in the active controller, it would be mean that there is a software bug. At this point, the standby controller will continue to act as the controller. Figure 33 demonstrates the switchover procedure. Introduction of BuDDI adds a latency of 0.2 ms, which is because our architecture is built on top of OpenVirtex that adds the delay to the control channel [77].

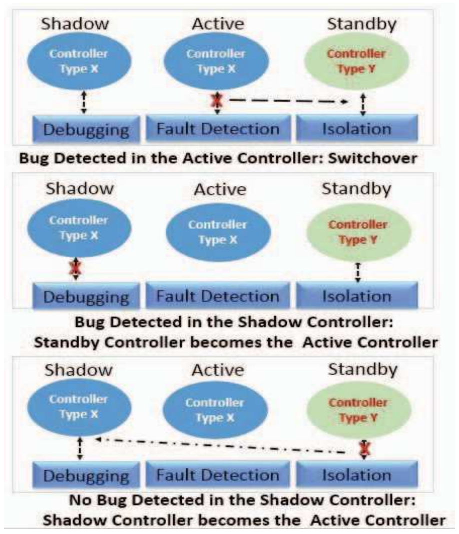


Figure 33 Shadow Controller Switchover (no bug detected vs bug detected)

5.3 Implementation and results

A. BuDDI Switchover Algorithm

Algorithm 1 shows the proposed BuDDI switchover mechanism. It covers how the switchover happens when any bug is detected and the convergence after the debugging process.

```
Algorithm 1: BUDDI Switchover
Result: Controller Switched Over
Error Detection module starts monitoring the Active Controller ;
while Bug Detected do
    Switchover Module starts switchover to Standby Controller as the new
    Active Controller and to Shadow Controller for automated Debugging;
    Transfer buggy state to the shadow controller; Transfer flow state to the
    standby controller;
    if Debugging Module finds same error in Shadow Controller then
        Continue using Standby Controller as  Active
        controller
    else
        Switchover Module switches over to Shadow Controller as the new
        Active  Controller;
        Transfer the flow state from Standby controller to
        Shadow controller
end
end
```

B. Simulation Setup and Results

In our initial experiment, we conducted switchover performance tests between heterogeneous and homogeneous con- trollers by using both Ryu and Pox controllers. Our test environment is a regular virtual machine with 2 GB RAM and an Ubuntu 14.04 operating system. We use Mininet [84] to simulate different network sizes (3, 8, and 11 switches) with linear topology. Each experiment has ping traffic, and was conducted 15 times, and the average values were taken.

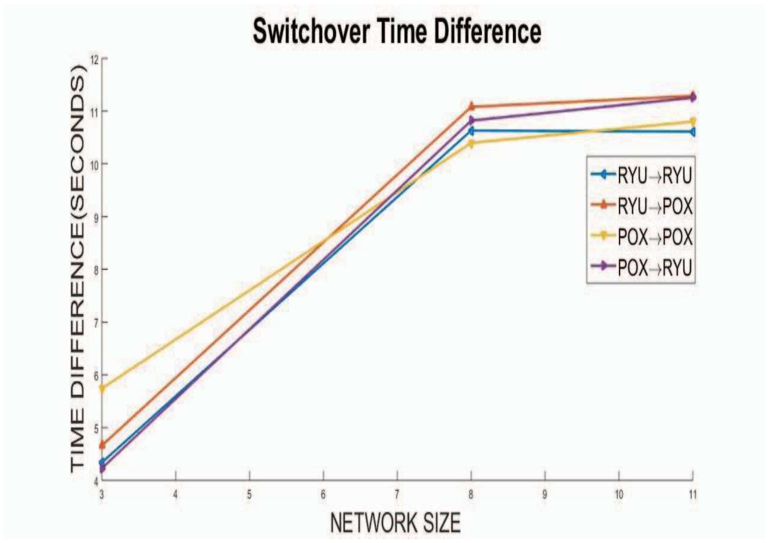


Figure 34 Heterogeneous vs Homogeneous Controllers Switchover Time

Figure 34 compares the switchover time for homogeneous and heterogeneous controllers in different simulated network sizes. Figure 35 presents two switchover time differences: (i) the switchover time difference between heterogeneous controllers, Ryu to Pox, and homogeneous controllers, Ryu -to -Ryu (a red line); and (ii) the switchover time difference between heterogeneous controllers, Pox to Ryu, and homogeneous controllers, Pox to Pox, (a blue line). The graph demonstrates that there is very little switchover time difference between the heterogeneous and homogeneous controller switchovers for the example networks we tested.

As we can see from the graph, when the network size in- creases, the switchover time increases as well. The switchover time is application specific. The architecture has been tested with a layer 2 simple switch. Since our test environment is a regular virtual machine, there is very minimum overhead to run a shadow controller. The result shows promise that BuDDI can effectively support a CMFD module switchover with minimal overhead.

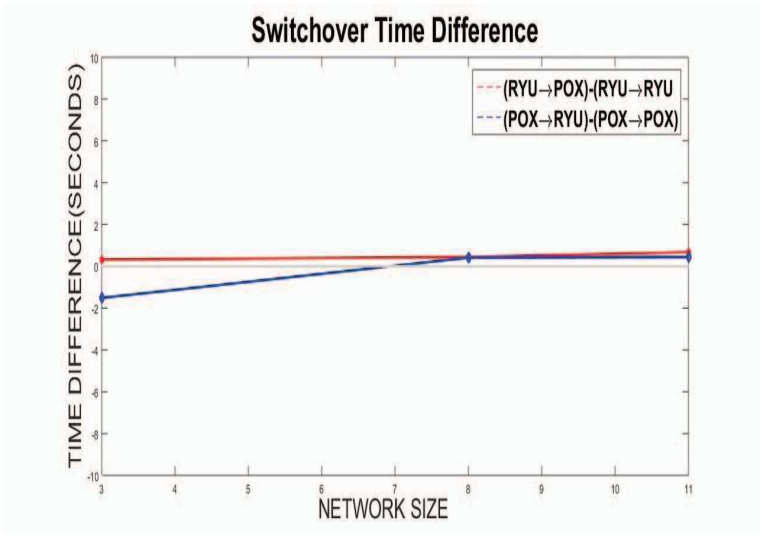


Figure 35 Heterogeneous vs Homogeneous Controllers Switchover Time Com- parison

5.4 Conclusion

We proposed a novel online software bug detection, de- bugging, and isolation (BuDDI) middlebox architecture for software-defined network controllers. Unlike the traditional recovery solutions, the proposed solution facilitates on-line based quality assurance, prediction, debugging, and, especially, common cause software failure mode resolutions by using the existing controllers on the top of the open source clustering facility, OpenVirtex.

We verified that BuDDI supports our claim of a heterogeneous controller switchover without causing additional performance overhead. By using the BuDDI algorithms and protocols in future work, we will further investigate additional debugging features and design an automated compatibility matrix over other existing controllers. We also plan to design a facility to transfer the buggy states and modules instead of switching over the entire controllers. We will further test our system by injecting SDN errors both from the software and network. We will also consider a wide range of applications and a high load on the network.

CHAPTER 6

RELIABILITY COST MODEL

Although the network system operation and management become complex to cope with modern network virtualization, softwareization, and federation approaches, the network reliability management still uses traditional availability measures such as Mean Time Between Failures (MTBF) and Mean Time Between Failures (MTTR), and a network Service Level Agreement (SLA) which is a time- based contract but hard to use or enforce in real world. Hence, the network outage measurement is an integral part of ensuring SLA conformance. However, when there is an outage, the current simple time-based SLA becomes overambitious for the customers to interpret the practical service impact due to the lack of suitable measurement tools and models. This paper proposes a novel network reliability cost model to ensure that the SLA covers customer service impact and damage. We classify network outages and calculate their effect on the network services to formulate a cost- based model, then we use Markov chain to capture the redundancy switchover and network outage rippling effect details when calculate downtime. Besides, we have performed evaluations using various campus network outage scenarios. The proposed cost-based model enables customers to identify the service impact of unplanned network outages to their networks instead of entirely depending on the service provider's data.

6.1 Reliability Modeling Issues

Internet is a massive, intricately connected network and requires high-skilled personnel to (re)configure and (re)install devices due to the complexity. It is expensive to add, remove, or move network appliances such as switches, routers, etc. The changes also have cascading impacts on network middleboxes and software functions such as Access Control Lists (ACLs), Virtual Local Area Networks (VLANs), and some other network domains. Additionally, network federation may cause a simple node or link failure to propagate and become global issues. For example, in 2020, COVID-19 caused a worldwide social lockdown that made an abrupt shift away from traditional business methods (ontact) to remote and online work as a new normal (untact). Many new untact technologies (i.e., Zoom, Microsoft Teams) have become more dependent on network availability. While this improves how businesses operate, it opens the possibility of risk from the rippling effect felt globally from a simple network outage. Cascading failure is the usual mechanism by which failures propagate to cause more massive impact and occur commonly in congested complex networks in the form of congestion generation, diffusion, and dissipation. Besides, network outages are experienced differently from site by site and from time to time. Moreover, the impact spectrum (both in time and space) of the network outages become much more significant than before. For example, there was a campus network outage in the University of Missouri KansasCity (UMKC) on Oct 22, 2020, at 06:54 CDT [85]. The outage has impacted both the UMKC email and Canvas system partially. As it has happened in early hour and all services are up and running in 4 hours by Oct 22, 2020, 10:49 CDT. However, unlike the traditional school days before COVID-19, the school had to cancel all the scheduled classes, meetings, and exams on the day.

A network Service Level Agreement (SLA) [89] is a time-based contract between a service provider and a customer, which defines service requirements and the expected service quality. Although the SLA definitions vary in vendors, services, and industries, providers typically promise quantifiable service quality to ensure High Availability (HA) network operations and services, which is better than five nines (99.999%) of network availability, according to equation, where Mean Time To Failures (MTTF) is a measured network outage, Mean Time To Repair (MTTR) is an estimated repair time, and Mean Time Between Failures (MTBF) is MTTF plus MTTR.

$$\begin{aligned} Availability(A) &= MTTF/(MTTF + MTTR) \\ &= MTTF/MTBF \end{aligned}$$

Hence, network outage measurement is an integral part of assuring SLA conformance. However, due to network virtualization, softwareization, multitenancy, and federation, modern network systems become complex and highly intricated. The service impact is not always proportional to the length of outage time, but outage affects differently. Hence, when there is an outage, the time-based SLA becomes overambitious for the customers to interpret the practical service impact due to the lack of suitable measurement tools and models. In most cases, these contracts are one-sided, with the service provider making a promise to the end-user (customer) the level of service it plans to provide. The customer has no measure to ensure if the service meets the promised SLA.

For example, Amazon had a service disruption in the Northern Virginia (US-EAST-1) Region on February 28th, 2017. The disruption impacted a subsystem necessary to serve all GET, LIST, PUT, and DELETE requests. This outage affected 148,213 websites and 121,761 unique domains, including the UMKC. The UMKC campus's Blackboard and many AWS-based services were not available for a couple of days. However, SLA was not impacted to the UMKC by this disruption because most SLAs are time-based and have no service cost. On the contrary, The Wall Street Journal reported the outage cost companies in the S&P 500 index $150 million, according to Cyence Inc. Apica Inc., a website- monitoring company, said 54 of the Internet's top 100 retailers saw website performance slow by 20% or more" [9]. It is assumed that not many companies received much compensation due to the fixed and limited SLA agreement.

We present a measurement study to bring forth the various unplanned network outage issues faced on a university campus network. We discuss the type of the network outage and the impact to the SLA. In addition, we perform evaluations using different network performance analysis tools. This study will shed light on unplanned network outage issues, as these will become applicable with the increasing prevalence of large metro area wireless networks. It is important to understand what the cost of network planning, design and implementing is to estimate cost as accurately as possible when it comes to decisions. Such estimations will enable a trade-off between the required availability of the network and the associated cost. Equipment cost model is used to estimate capital expenditure (CapEx) costs. Operational expenditure (OpEx) costs is associated with continuous running of the network. CapEx mainly contributes to the xed infrastructure of a company, and they depreciate over time [24]. They include the purchase of land and buildings (e.g., to house the personnel), network infrastructure (e.g., optical ber and IP routers), and software (e.g., network management system). Buying equipment has always been considered part of CapEx, regardless of whether the payment is made all at once or spread over time. Additionally, interest paid for a loan is included in CapEx. Land and building which is composed of network land/building and personnel building contribute toward CapEx as illustrated by 1. OpEx represents the cost of keeping the company operational and include costs of technical and commercial operations, administration, etc. The majority contributors to OpEx for network service providers can be classified into three major categories: the portion directly related to continues cost of infrastructure, running the network and outage recovery. Operating an existing network (which has already been set up), equipment installation, and some general expenditure (aspects not specific to a network operator). Although, Miscellaneous OpEx such as payroll, talent management, cost of infrastructure heating, building cleaning and administration costs are not included in the figure, they contribute toward OpEx. In addition to OpEx and CapEx, Total Cost of Ownership (TOC) illustrated by Figure 36 includes service providers factor in service penetrations, population density per geographic area and service demand per user to determine the benefit of setting up network.

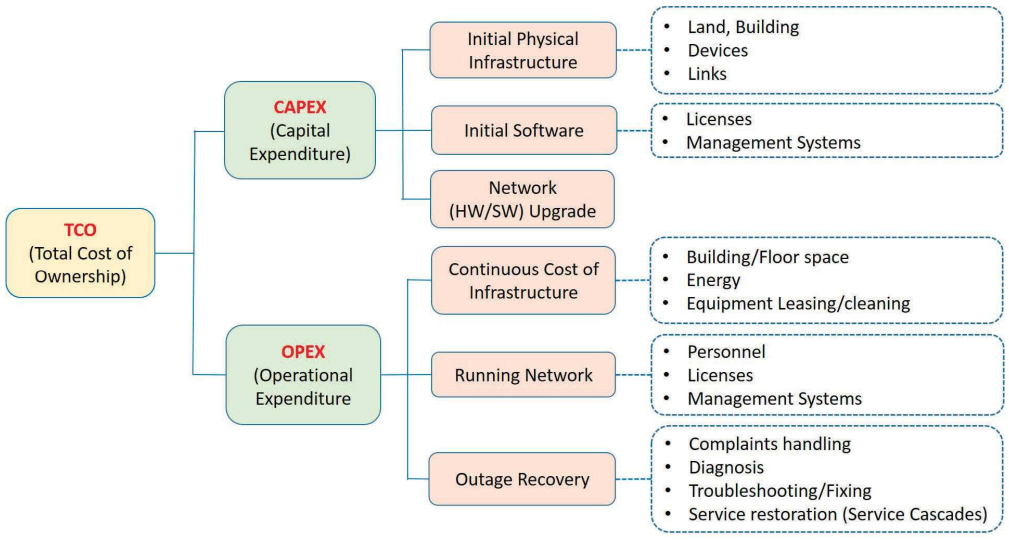


Figure 36 Total Cost of Ownership (TCO)

6.2 COST OF NETWORK OUTAGE

Network outage could impact both CapEx and OpEx. It can be classified into two major categories: planned network outage and unplanned network outage. Planned network outage mainly impacts continuous cost of structure and maintenance (device upgrade and cleaning) in OpEx. For example, Internet Service Providers (ISP) send out an announcement on their web page when they schedule a planned outage for upgrade or maintenance, informing their clients that service will not be available during the upgrade or maintenance window. However, with SDN, the cost of planned outage is minimal and customers should not experience any outage.

Most planned outages are provisioned in the Service Level Agreement (SLA) [89], a contract between a service provider and it's an internal or external customers that documents what services the provider will furnish, and defines the performance standards the provider is obligated to meet. Unplanned network outage can be grouped into three subcategories: human error (where a planned outage goes wrong), natural disaster (such as thunder storm, excessive heat leading to electrical fire, etc.) or device failure (where unsupported devices get introduced to the network causing network failure). For example, an unplanned outage could be triggered by human error from mis-configuration while doing planned outage or from unplugging the wrong device, etc. Figure 37 illustrates effects of network outage.

SDN makes several sets of attributes that minimize the impact on OpEx. First, mechanized and automated roll-out (creation/provisioning/ removal/termination) of capacity of SDN functions including transport connections, based on near-realtime demand, application performance, and so on, are enabled by features such as network programmability and open APIs. This enables automated elasticity for fast deployment of network service roll-out [100]. Furthermore, SDN removes the dependency between software and hardware. Software service can be deployed on any Hardware. SDN also enables multi-tenancy and resource pooling for multiple software functions on the same hardware. Hypervisors and associated management and orchestration software facilitate virtualization of the network functions and the automation of network processes are attributes found in SDN, lacking from physical network. SDN consolidates and optimizes Service agility through enabled service abstraction and automation. Operating model change by blurring staff responsibility of network and IT [100].

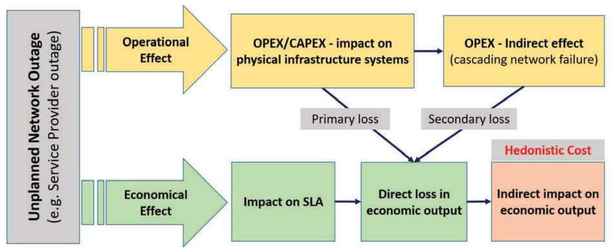

Unplanned Network Outage
(e.g. Service Provider outage)
Operational Effect
OPEX/CAPEX - impact on physical infrastructure systems
OPEX - Indirect effect (cascading network failure)
Primary loss
Secondary loss
Hedonistic Cost
Economical Effect
Impact on SLA
Direct loss in economic output
Indirect impact on economic output

Figure 37: Effect of Unplanned Outage

A. Root Cause of Unplanned Outage

There is a major difference between planed and unplanned network outage. With planned outage, applications and servers are brought down gracefully in preparation to the outage with little need for cleanup and restoration. Unplanned outage depending on the impact could be as low as a glitch that unnoticed by end users to high unrecoverable network. Human error is the second highest root cause of unplanned failure in a data-center reported in 2016 [101]. Figure 38 illustrated the different root cause of unplanned data-center outage.

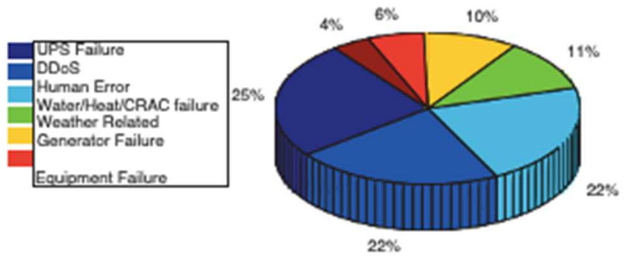


Figure 38: Root Cause of Unplanned Data-center Outage in 2016

B. Cost of Reliability

It can achieve the business system reliability by minimizing these four categories: recovery cost, revenue loss, productivity loss, and hedonistic cost. The impact of network failure varies according to the network dependency of the business [96]. For example, the questions include “Is revenue generated primarily on- line?”, “Is revenue highly dependent on the use of email, Learning Management System (LMS), cloud accessibility, databases, or other online resources?” and “When is the most significant business hour?” We have defined initial cost-based reliability variables in 4 and an additive formula of the total cost of outage (TC) with the help of the Department of Economics of the UMKC. The total cost of outage (TC) includes Impact Factor (α), Recovery Cost (RC), Revenue Loss (RL), Production Loss (PL), and Hedonistic Loss (HL).

Variable	Description
α	% reliant on network up-time
E	Number of employee
HW	Number of Hours worked
W	Median Wage / Hour
GR	Gross Revenue Generated / Year
TH	Total Business Hour / Year
HD	Hours of Downtime
S	Cost of service to recover
L	% Potential loss to competitor

Table 11: Cost-based Reliability Variables

1) Impact Factor (α): Impact factor (α) is % the reliant on up-time as illustrated by Table 11.

α (n) = % of dependent on up − time

2) Recovery Cost (RC): Recovery cost depend on the impact factor(α), number of employee(E) and wage (W), hours to recover (HW) and the cost of service (such as device, tool, date etc.) to recover (S).

$RC\ (n) = \alpha \times (E \times W) \times (HW) + S$

3) Revenue Loss (RL): Revenue Loss is affected by the impact factor and average revenue generated per day.

$$RL\,(n) = \alpha \times (\frac{GR}{TH}) \times HD$$

4) Production Loss (PL): Production Loss is affected by the number of

employees, the average wage, hour of downtime and impact factor.

PL (n)= (NE × W) × HD × α

5) Hedonistic Loss (HL): Hedonistic Loss includes all intangible loss impacted by the total sell, impact factor and % lost potential business to competitors. For example, customers leaving because they are frustrated by the network outage.

$$HL\,(n) = (\frac{R}{\alpha}) \times L$$

6) Total Cost of Outage (TC): Total cost of outage include all the above.

TC (n) = RC + RL + PL + HL

From the above formulars, we can see the parameter HD, which stands for Hour of Downtime, play an important role in the overall cost of reliability. However, as we described before, the after outage behavior, including switchover to redundant system, outage rippling effect etc. need to be accurately captured to represent the overall reliability experience people experienced in the real world.

6.3 Markov Model

As The Markov modeling approach is an accurate availability model in capturing the dependencies. For example, Markov chain can express the redundancy switchover procedure explicitly with states and state transitions.

6.3.1 1 Reliability Characteristics in model

Our The following reliability characteristics are taken into consideration:

- MTBF – The mean time between failure of an single unit
- MTTR – The mean time to repair of an single unit
- Switchover – The operation of switching communication traffic from one unit to another unit

- Switchover Time – The time takes for the communication traffic to switch from one unit to another unit
- Switchover Coverage – The probability of the successful communication traffic switchover
- M-of-N Redundancy – For a system to be fully functional, it needs minimum M units to work out of the total N units. If M-1 units are working, 1/M of the traffic is affected. If fewer units are working, more communication traffic is affected proportionally.
- Active Unit – In normal condition, unit actively takes traffic
- Standby Unit – In normal condition, unit does not take traffic
- M-of-N Active-Standby Redundancy – In normal condition, M units are active and N-M units are standby. When active units fail, standby units take over to be active. The communication traffic is not affected with M active units working and taking traffic.

- M-of-N Load-Sharing Redundancy – In normal condition, N units are all active and each takes 1/N of the traffic. When units fail, the rest of the units each take more traffic. The communication traffic is not affected with at least M units working and taking traffic.

6.3.2 Markov Model for reliability system

Markov model has the capability to capture the time and sequence dependency. It has been used to model the redundant system unplanned hardware and software failures behavior as well as planned software upgrades.

6.3.2.1 M-of-N Load-sharing Markov Model

Markov chain for M-of-N load-sharing redundant system is shown in Figure 39.

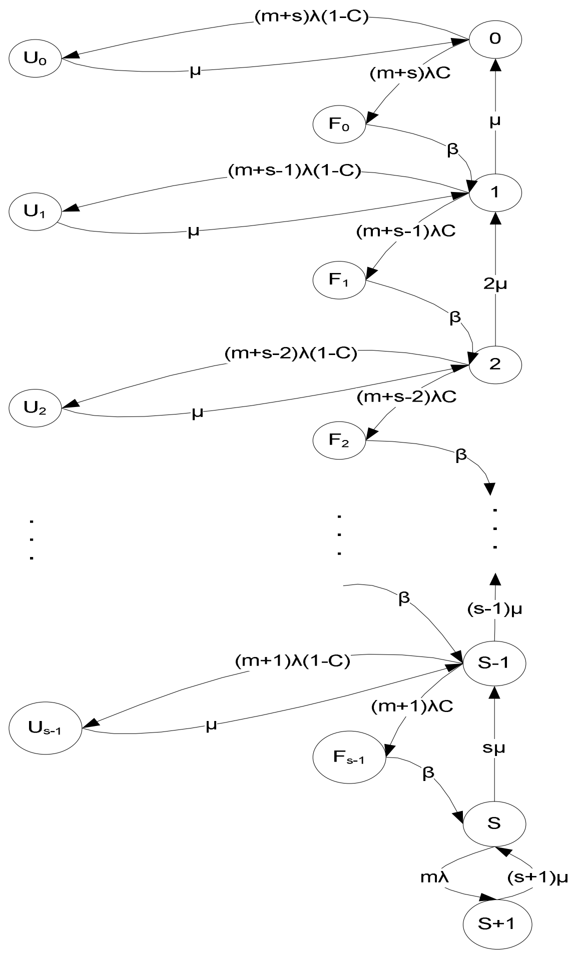


Figure 39 M-of-N Load-sharing Markov Chain

Legend

m	Equals M in M-of-N system. Is minimum number of units needed for system to work without affecting communication traffic
s	Equals (N-M) in M-of-N system. Is maximum number of spare units which can fail without affecting communication traffic
N	Total units
M	Is minimum number of units needed for system to work without affecting communication traffic

λ	Failure rate of individual unit
μ	Repair rate of individual unit
β	Rate if traffic switchover from failed unit to other healthy unit
C	Switchover coverage factor

State Descriptions

0	All N=m+s units working
U0	One unit failed, undetected
F0	One unit failed, detected
1	One of the s spares has been used and traffic has been taken over from the one originally failed unit.
U1	Second unit failed, undetected
F1	Second unit failed, detected
2	Two of the m spares has been used and traffic has been taken over from the two originally failed units.
S	All s spares have been used and traffic has been taken over from the originally failed s units.
S+1	s+1 units failed. System can not take full traffic any more. 1/M system capacity is lost

In Figure 39, state 0 is the normal state when everything is working. The entire N=m+s units are working in the load-sharing mode, each of them take 1/N of the traffic. When there is a detected failure in one of the units, the system goes to state F0, which triggers the switchover operation. The rate of entering state F0 is (m+s)λC because there are (m+s) units that can fail, each of them fails with rate λ, and C is the probability of detecting this failure. After a switchover delay represented by the transition rate β, the system goes into state 1 which represents (m+s-1) working units with 1 spared unit being used. With rate μ that failed unit is repaired and the system returns to state 0. This is the normal flow of the system when failures happen. The transition from state 0 to state U0 represents the case where there is a failure in a unit but it is not detected. The rate of going into state U0 is (m+s)λ(1–C). The repair rate from state U0 is also μ.

The system will stay in state 1 after the switchover, then it will have a rate of (m+s-1)λC and (m+s-1)λ(1-C) to go into state F1 and U1, based on whether another unit fails covered or uncovered this time. The same state transition process will continue, as long as the there are still spares available.

Finally, in state S, all the possible spare units have been used. There are M units that take the traffic, and the system is still working as the minimum required M units are still fine. If there is one more unit fail, the system will transit to S+1 state and lose 1/M of the traffic. As expected, the probability of this happening could be small and it will be reflected in the time the system spends in state S+1.

The working states are 0, 1, 2 to S. The failure states are S+1, U0 to Um-1 and F0 to Fm-1.

6.3.2.2 M-of-N Active-standby Markov Model

Markov chain for M-of-N active-standby redundant system is shown in Figure 40.

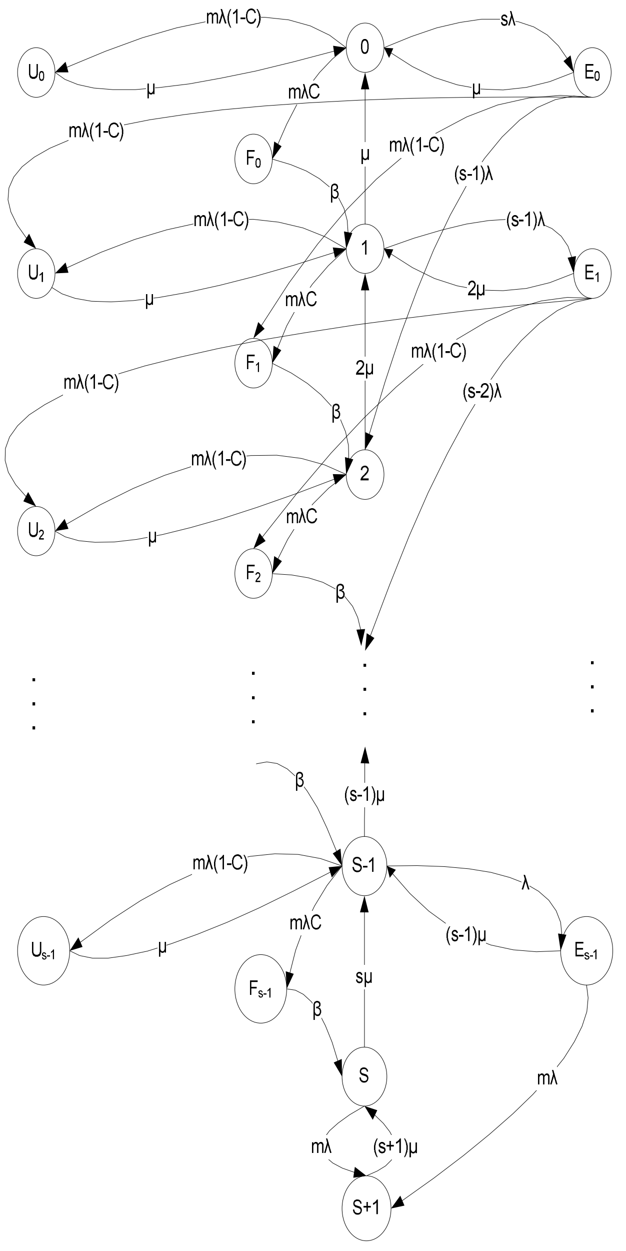


Figure 40 M-of-N Active-standby Markov Chain

Legend

m	Equals M active units in M-of-N system. Is minimum number of units needed for system to work without affecting communication traffic
s	Equals (N-M) standby units in M-of-N system. Is maximum number of spare units which can fail without affecting communication traffic
N	Total units
M	Is minimum number of units needed for system to work without affecting

communication traffic

λ	Failure rate of individual unit
μ	Repair rate of individual unit
β	Rate if traffic switchover from failed unit to other healthy unit
C	Switchover coverage factor

State Descriptions

0	All m active units are working and taking traffic. All s standby units are working and ready to take over the traffic if any active unit fails
U0	One active unit failed, undetected
F0	One active unit failed, detected
E0	One standby unit failed
1	One of the m standby units has been used and traffic has been taken over from the one original failed active unit.
U1	Second active unit failed, undetected
F1	Second active unit failed, detected
S1	Second standby unit failed
2	Two of the m standby unit has been used and traffic has been taken over from the two original failed units.
S	All s standby units have been used and traffic has been taken over from the s failed active units.
S+1	s+1 units failed. System can not take full traffic any more. 1/M system capacity is lost.

In Figure 40, state 0 is the normal state when everything is working. All the m units are working and actively taking traffic. s units in the standby mode. When there is a detected failure in one of the m active units, the system goes to state F0, which triggers the switchover operation. The rate of entering state F0 is mλC because there are m active units that can fail, each of them fails with rate λ, and C is the probability of detecting this failure. After a switchover delay represented by the transition rate β, the system goes into state 1 which represents m active units with 1 standby unit being used. With rate μ that failed unit is repaired and the system returns to state 0. This is the normal flow of the system when active unit failures happen. The transition from state 0 to state U0 represents the case where there is a failure in an active unit but it is not detected. The rate of going into state U0 is mλ(1–C). The repair rate from state U0 is also μ. When a standby unit fails in state 0, the system goes to E0. While the systems stay at E0, if three is a second active unit fails and detected, it goes to F1; if the second active unit fails but undetected, it goes to U1; if second standby fails, it goes to state 2.

The system will stay in state 1 after the switchover, then it will have a rate mλC and mλ(1-C) of going into state F1 and U1, based on whether another of the active units fail covered or uncovered this time. When another standby unit fails, the system goes to state E1 with a rate (s-1)λ. The same state transition process will continue, as long as the spares are not used up.

Finally, in state S, all the possible standby units have been used. There are M units are taking the traffic and the system is still working as the minimum required M units are still fine. If there is one more unit fail, the system will transit to S+1 state and lose 1/M of the traffic. As expected, the probability of this happening could be small and it will be reflected in the time the system spends in state S+1.

The working states are 0, 1, 2 to S. The failure states are S+1, U0 to Um-1 and F0 to Fm-1.

6.3.2.3 Planned Software Upgrade Markov Model

Markov chain for planned software upgrade is shown in Figure 41.

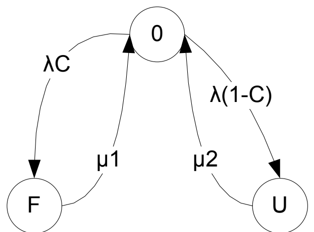


Figure 41 Software Upgrade Markov Chain

Legend

λ	Software upgrade rate
μ1	Repair rate of successful software upgrade
μ2	Repair rate of unsuccessful software upgrade
C	Software upgrade coverage factor
	State Descriptions
0	Software working
F	Successful software upgrade by switchover
U	Failed software upgrade, reset the system

In Figure 41, state 0 is the normal state when software is working. λ represents the software upgrade rate, such as twice a year. To save the downtime, software upgrade is normally done by saving a new version of the software in a standby unit, and the upgrade operation is done by a switchover. The successful switchover case is represented in F state and the unsuccessful one is represented in U state, which normally means a system reset is needed. Repair rates μ1 represent how quickly the switchover can be and repair rate μ2 represents how quickly the system reset can be.

6.4. Verification with Reliability and Cost Analysis

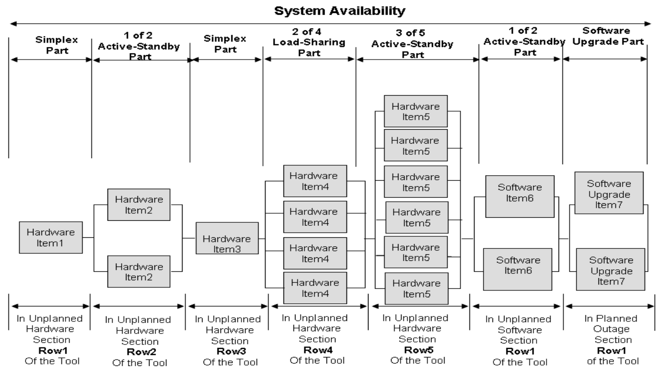
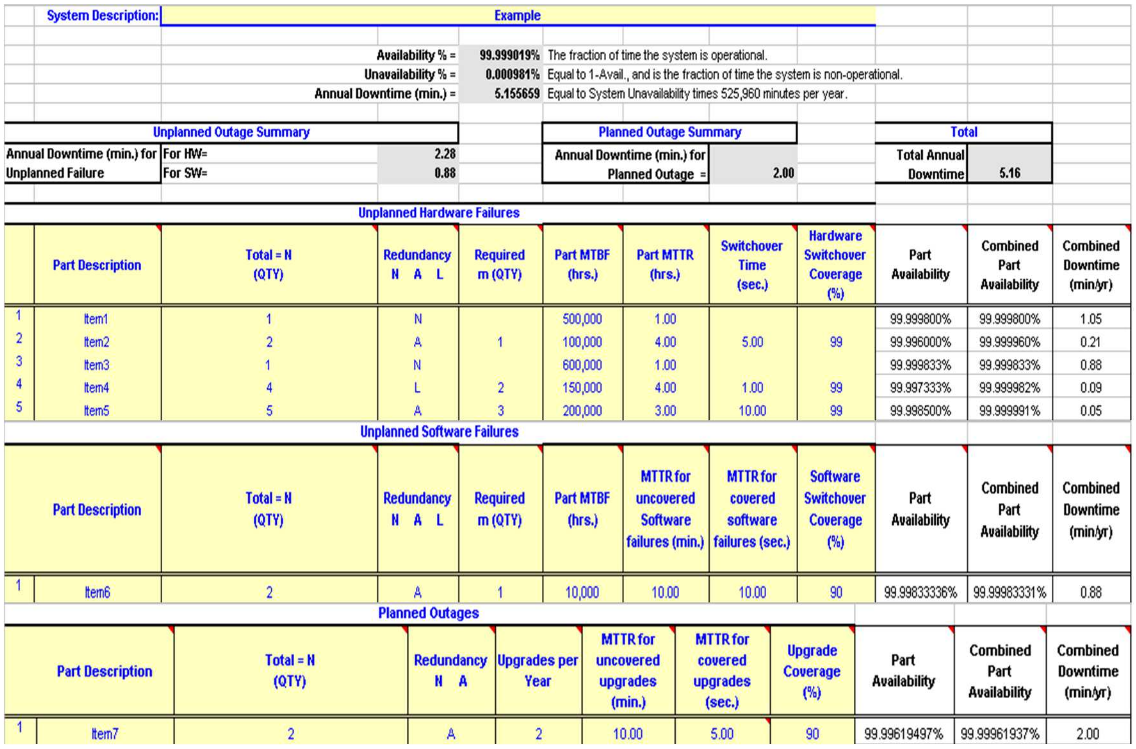

System Availability
Simplex Part
1 of 2 Active-Standby Part
Simplex Part
2 of 4 Load-Sharing Part
3 of 5 Active-Standby Part
1 of 2 Active-Standby Part
Software Upgrade Part
Hardware Item1
Hardware Item2
Hardware Item2
Hardware Item3
Hardware Item4
Hardware Item5
Software Item6
Software Upgrade Item7
In Unplanned Hardware Section Row1 Of the Tool
In Unplanned Hardware Section Row2 Of the Tool
In Unplanned Hardware Section Row3 Of the Tool
In Unplanned Hardware Section Row4 Of the Tool
In Unplanned Hardware Section Row5 Of the Tool
In Unplanned Software Section Row1 Of the Tool
In Planned Outage Section Row1 of the Tool

Figure 42 Reliability Block Diagram

Reliability Block Diagram is commonly used in the industry. Engineers typically use the serial and parallel RBD to represent their system. In this type of RBD, the availability of a system is characterized by a set of serial and parallel blocks interconnected. [1] For an IP networking system the blocks need to include items in hardware and software unplanned failures as well as software planned upgrades. A RBD example is illustrated in Figure 42, where simplex blocks in series represent non-redundant parts and blocks in parallel represent parts in redundancy. Item1 is a simplex hardware part, Item2 is 1-of-2 active-standby redundant hardware part. Item3 is simplex hardware part again. Item4 is in 2-of-4 load-sharing redundancy while item5 is 3-of-5 active-standby redundant parts. The software failure is represented in item6 for 1-of-2 active-standby redundancy. Finally, the planned software upgrade outages are represented as redundant software upgrade parts in item7.

The availability analysis tool is used after the development of system RBD. The blocks in RBD are translated into the rows in the excel-format analysis tool.



System Description: Example

Availability % =	99.999019%	The fraction of time the system is operational.
Unavailability % =	0.000981%	Equal to 1-Avail., and is the fraction of time the system is non-operational.
Annual Downtime (min.) =	5.155659	Equal to System Unavailability times 525,960 minutes per year.

Unplanned Outage Summary

Annual Downtime (min.) for Unplanned Failure	For HW=	2.28
	For SW=	0.88

Planned Outage Summary

Annual Downtime (min.) for Planned Outage =	2.00

Total

Total Annual Downtime	5.16

Unplanned Hardware Failures

	Part Description	Total = N (QTY)	Redundancy N A L	Required m (QTY)	Part MTBF (hrs.)	Part MTTR (hrs.)	Switchover Time (sec.)	Hardware Switchover Coverage (%)	Part Availability	Combined Part Availability	Combined Downtime (min/yr)
1	Item1	1	N		500,000	1.00			99.999800%	99.999800%	1.05
2	Item2	2	A	1	100,000	4.00	5.00	99	99.996000%	99.999960%	0.21
3	Item3	1	N		600,000	1.00			99.999833%	99.999833%	0.88
4	Item4	4	L	2	150,000	4.00	1.00	99	99.997333%	99.999982%	0.09
5	Item5	5	A	3	200,000	3.00	10.00	99	99.998500%	99.999991%	0.05

Unplanned Software Failures

	Part Description	Total = N (QTY)	Redundancy N A L	Required m (QTY)	Part MTBF (hrs.)	MTTR for uncovered Software failures (min.)	MTTR for covered software failures (sec.)	Software Switchover Coverage (%)	Part Availability	Combined Part Availability	Combined Downtime (min/yr)
1	Item6	2	A	1	10,000	10.00	10.00	90	99.99833336%	99.99983331%	0.88

Planned Outages

	Part Description	Total = N (QTY)	Redundancy N A	Upgrades per Year	MTTR for uncovered upgrades (min.)	MTTR for covered upgrades (sec.)	Upgrade Coverage (%)	Part Availability	Combined Part Availability	Combined Downtime (min/yr)
1	Item7	2	A	2	10.00	5.00	90	99.99619497%	99.99961937%	2.00

Figure 43 Reliability and Availability Analysis Tool User Interface

The user interface of the availability analysis tool is shown in Figure 43. It is comprised of three portions, i.e., input area for part availability calculation (left half of the data table), result area from part availability calculation (right half of the data table), and result area for system availability calculation (above the data table).

The worksheet parameters and metrics are defined as follows.

Part Calculation Inputs

Part Description—describes the part under availability calculation.

Total Part Quantity N—specifies the total number of parts for a given part type. In case of M-of-N redundancy, it includes all the N units.

Redundancy—describes redundancy types with three selectable character values: “N”, “A”, and “L”.

“N” indicates that no redundancy exists.

“A” indicates active-standby redundancy. By the active-standby redundancy, M units are active and N-M unit serves as a standby unit. When any one of the M active units fails, the standby unit takes over.

“L” indicates load-sharing redundancy. With the load-sharing redundancy, all N units are active and carrying traffic in roughly equal 1/N distributions. If one unit fails, the other N-1 units take over the load.

Required Part Quantity m—specifies the required number of parts for a given part type. In case of M-of-N redundancy, it is the minimum required M units.

Part MTBF (hours)—is the MTBF of a single unit.

Part MTTR (hours)—is the MTTR of a single unit.

Switchover Time (sec.)—is the amount of time spent on switching from one (failed) unit to another (healthy) unit.

Switchover Coverage Factor (%)—is the probability that a failure in the working unit can be successfully detected and a switchover from the failed unit to a healthy unit is successful. It depends on the fault detection on the working part; the successful operation of the switchover mechanism, and the readiness of spare part on demand.

Part Calculation Results

Part Availability (%)—is the calculated availability for a given single unit.

Combined Part Availability (%)—is the calculated availability for a set of redundant part units. If there is no part redundancy, it is equal to the Part Availability.

Combined Part Downtime (min./year)—is the calculated annual downtime for a set of redundant part units.

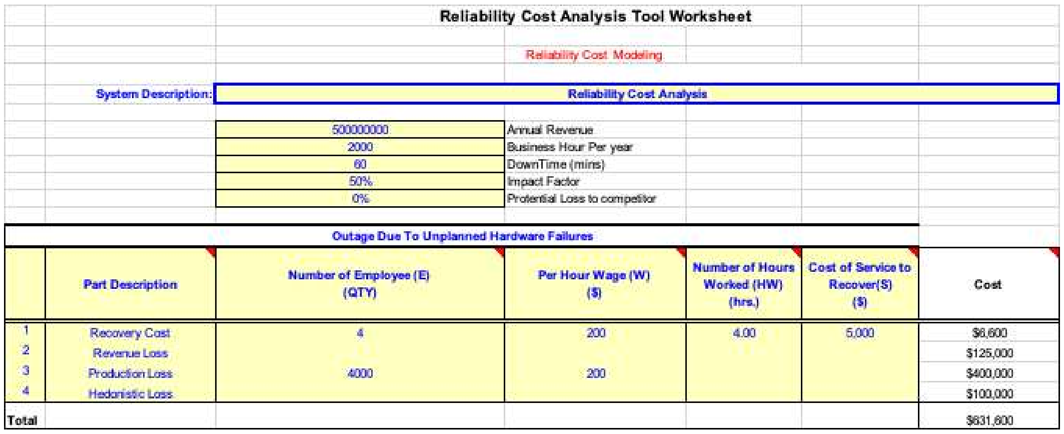
System Calculation Results

System Availability (%)—is the calculated availability for the entire system.

System Unavailability (%)—is the calculated unavailability for the entire system.

System Annual Downtime (min./year)—is the calculated annual downtime for the entire system.

As an example, Figure 42 shows a worksheet for the system described in Figure 41. In the worksheet, Row-1 records the availability data for a simplex hardware part Item1. Similarly, Row-2 entry captures a pair of 1-of-2 active-standby redundant parts Item2. Row-4 calculates the availability of 2-of-4 load-sharing Item4, where two items are used to backup the other two items. The calculation results are displayed on a per-part type basis in the data table.



Reliability Cost Analysis Tool Worksheet

Reliability Cost Modeling

System Description: **Reliability Cost Analysis**

500000000	Annual Revenue
2000	Business Hour Per year
60	DownTime (mins)
50%	Impact Factor
0%	Protential Loss to competitor

Outage Due To Unplanned Hardware Failures

	Part Description	Number of Employee (E) (QTY)	Per Hour Wage (W) ($)	Number of Hours Worked (HW) (hrs.)	Cost of Service to Recover(S) ($)	Cost
1	Recovery Cost	4	200	4.00	5,000	$6,600
2	Revenue Loss					$125,000
3	Production Loss	4000	200			$400,000
4	Hedonistic Loss					$100,000
Total						$631,600

Figure 44 Cost Model Tool

The user interface of the reliability cost tool is shown in Figure 44. We analyze network outages and calculate their effect on the network services to formulate a cost-based model including recovery cost, revenue loss, productivity loss, and hedonistic cost.

CHAPTER 7

SUMMARY AND FUTURE WORK

As our daily life gets more dependent on essential and important services connected to the Internet, network reliability has never been more important. To deal with network reliability, we have mainly focused and studied high availability and scalability. This dissertation specifically focused on Software-Defined Networks (SDN), identified new issues of high availability and scalability of SDN, and solved the problems using various schemes and algorithms.

Little attention has been paid to the SDN network reliability management that suffers from scalability and latency issues. We proposed a novel Network Architecture-aware Reliability Management Schemes to efficiently orchestrate different reliability monitoring mechanisms over SDN network architecture and synchronize the control messages among different controllers and applications. We enabled the platform to provide fast and smart decision making information for fast failure detection and recovery. A prototype is implemented on Cisco's OpenDayLight (ODL). Extensive experiment results exhibit that our algorithm achieves effective and efficient network failure detection while generating limited LLDP message overhead. We propose a novel network reliability cost model to ensure that the SLA covers customer service impact and damage.

By applying the proposed reliability scenarios and algorithms, as future work, further investigation and development can be continued for efficient and reliable mechanisms to achieve reliability for the carrier-grade SDN networks where we consider a large scale network deployment.

REFERENCE LIST

[1] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., and Parulkar, G. ONOS: Towards an Open, Distributed SDN OS. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (2014), HotSDN '14.

[2] Linux Bonding Driver How-To. https://www.kernel.org/doc/Documentation/networking/bonding.txt.

[3] Cai, Z., Cox, A. L., and Ng, T. E. Maestro: A System for Scalable OpenFlow Control. Tech. Rep. TR10-11, Rice University, Dec. 2010.

[4] Casado, M., Freedman, M., Pettit, J., Luo, J., McKeown, N., and Shenker, S. Ethane: Taking Control of the Enterprise. *Proceedings of ACM SIGCOMM Computer Communication Review 37* (2007), 1–12.

[5] Choi, T., Cho, C., Yoon, S., Yang, S., Park, H., and Song, S. DEMO: Unified Virtual Monitoring & Analysis Function over Multi-core Whitebox. In *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)* (Dec. 2014).

[6] Choi, T., Kang, S., Yoon, S., Yang, S., Song, S., and Park, H. SuVMF: Software-defined Unified Virtual Monitoring Function for SDN-based Large-scale Networks. In *Proceedings of The 9th International Conference on Future Internet Technologies (CFI)* (Jun. 2014).

[7] Choi, T., Lee, B., Kang, S., Song, S., Park, H., Yoon, S., and Yang, S. IRIS-CoMan: Scalable and Reliable Control and Management Architecture for SDN-enabled Large-scale Networks. *Journal of the Network and Systems Management 23* (2015), 252–279.

[8] Choi, T., Song, S., Park, H., Yoon, S., and Yang, S. SUMA: Software-defined Unified Monitoring Agent for SDN. In *Proceedings of IEEE Network Operations and Management Symposium (NOMS)* (May 2014).

[9] The Art of Application-Centric Networking. http://www.cisco.com/en/US/solutions/collateral/ns1015/ns175/ns348/ns1126/cisco_td_030513_fin.pdf.

[10] Designing a Campus Network for High Availability. http://www.cisco.com/application/pdf/en/us/guest/netsol/ns432/c649/cdccont_0900aecd801a8a2d.pdf.

[11] Beacon. https://floodlight.atlassian.net/wiki/spaces/Beacon/overview.

[12] Big Network Controller. http://bigswitch.com/products/SDN-Controller.

[13] Project Floodlight. https://github.com/floodlight.

[14] IRIS: The Recursive SDN OpenFlow Controller by ETRI. http://openiris.etri.re.kr/.

[15] Open Mul: High performance SDN. https://www.kulcloud.com/beemopenmul/.

[16] NOX. https://github.com/noxrepo/nox.

[17] OpenDaylight. http://www.opendaylight.org/.

[18] POX. https://OpenFlow.stanford.edu/display/ONL/POX+Wiki.

[19] Ryu SDN framework. https://github.com/faucetsdn/ryu.

[20] Trema. https://trema.github.io/trema/.

[21] curl Man page. http://curl.haxx.se/docs/manpage.html.

[22] Curtis, A., Mogul, J., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. DevoFlow: Scaling Flow Management for High-performance Networks. In *Proceedings of ACM SIGCOMM* (Aug. 2011), pp. 254–265.

[23] Desai, M., and Nandagopal, T. Coping with Link Failures in Centralized Control Plane Architecture. In *Proceedings of IEEE COMmunication Systems and NET- works (COMSNET)* (2010), pp. 79–88.

[24] Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., and Kompella, R. Towards an Elastic Distributed SDN Controller. In *Proceedings of ACM SIGCOMM Workshop on HotSDN* (2013).

https://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p7.pdf

[25] Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., and Kompella, R. ElastiCon: An Elastic Distributed SDN Controller. In *Proceedings of ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)* (2014). https://ieeexplore.ieee.org/document/7856398

[26] Element Management System (EMS) network manager. http://www.sonus.net/ node/96.

[27] Project Floodlight: Module Applications. https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343509/Modul e+Applications

[28] Floodlight RESTful API. https://floodlight.atlassian.net/wiki/display/floodlight con- troller/Floodlight+REST+API.

[29] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. NOX: Towards an Operating System for Networks. In *SIGCOMM Computer Communication Review* (Jul. 2008), vol. 38, pp. 105–110.

[30] Ethernet Automatic Protection Switching (EAPS). https://datatracker.ietf.org/doc/html/rfc3619, Oct. 2003.

[31] IP Multicast Load Splitting - Equal Cost Multipath (ECMP). https://www.cisco.com/c/en/us/td/docs/ios/12_4t/ip_mcast/configuration/guide/ mctlsplt.html

[32] Ethernet Ring Protection Switching(ERPS). http://www.cisco.com/c/en/us/td/docs/ ios-xml/ios/cether/configuration/xe-3s/ce-xe-3s-book/ce-g8032-ering-pro.html.

[33] EtherChannels. http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3550/ software/release/12-1_13_ea1/configuration/guide/3550scg/swethchl.html.

[34] Fast Re-Routing (FRR). http://tools.ietf.org/html/rfc4090, May 2005.

[35] Cisco Hot Standby Router Protocol (HSRP). https:// www.ietf.org/rfc/rfc2281.txt,

Mar. 1998.

[36] Link Aggregation Control Protocol (LACP), https://www.cisco.com/c/en/us/td/docs/ios/12_2sb/feature/guide/gigeth.html, Mar. 2007.

[37] Graceful OSPF Restart: Non-Stop Forwarding (NSF). https://datatracker.ietf.org/doc/html/rfc3623, Nov. 2003.

[38] Non-Stop Routing (NSR). https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_ospf/configuration/xe-3s/iro-xe-3s-book/iro-nsr-ospf.pdf.

[39] Resilient Packet Ring (RPR). http://www.ieee802.org/17/documents.htm.

[40] Stateful Switch-Over (SSO). https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ha/configuration/15-sy/ha-15-sy-book/ha-config-stateful-switchover.html

[41] Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6. http://tools.ietf.org/html/rfc5798, Mar. 2010.

[42] Heller, B., Sherwood, R., and McKeown, N. The Controller Placement Problem. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)* (2012), pp. 7–12.

[43] JSON Data Interchange Standard. http://json.org/.

[44] GitHub: Open source parser Jsoncpp. https://github.com/open-source-parsers/jsoncpp.

[45] Kandula, S., Sengupta, S., Greenberg, A., and Patel, P. The Nature of Datacenter Traffic: Measurements and Analysis. In *Proceedings of ACM IMC* (2009), pp. 202–208.

[46] Keepalived: Load Balancing and High-Availability. http://www.keepalived.org/.

[47] Kempf, J., Bellagamba, E., Kern, A., Jocha, D., Takacs, A., and Skoldstrom, P. Scalable Fault Management for OpenFlow. In *Proceedings of IEEE International Conference on Communications (ICC)* (2012), pp. 6606–6610.

[48] Kim, D., Park, J.-W., Song, S., Choi, B.-Y., Park, H., Paik, E.-K., Jeong, K.-T., and Hong, S. Method and Apparatus for Processing a Control Message in Software-Defined Network. *Korean Patent*, 1020130143244 (Nov. 2013).

[49] Kim, H., Santos, J., Turner, Y., Schlansker, M., Tourrilhes, J., and Feamster, N. CORONET: Fault Tolerance for Software Defined Networks. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)* (2012). https://www.princeton.edu/~hyojoonk/publication/coronet_icnp12.pdf

[50] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., and Shenker, S. ONIX: A Distributed Control Platform for Large-scale Production Networks. In *Proceedings of USENIX conference on Operating Systems Design and Implementation (OSDI)* (2010). https://www.usenix.org/legacy/events/osdi10/tech/full_papers/Koponen.pdf

[51] Krishnamurthy, A., Chandrabose, S. P., and Gember-Jacobson, A. Pratyaastha: An Efficient Elastic Distributed SDN Control Plane. In *Proceedings of the Workshop*

on Hot Topics in Software Defined Networking (HotSDN) (2014), pp. 133–138.

[52] Kuźniar, M., Perešíni, P., Vasić, N., Canini, M., and Kostić, D. Automatic Failure Recovery for Software-defined Networks. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)* (2013), pp. 159–160.

[53] Lee, K., Jang, I., Shin, M., and Baek, S. Design of Super Controller for Large Scale Software-Defined Networks. In *OSIA Standards & Technology Review* (Sep. 2012).

[54] Pantou: OpenFlow 1.0 implementation for OpenWRT. http://imgonamakeublackoutproductions.blogspot.com/2014/10/ztpantou-openflow-10-for-openwrt.html

[55] Linksys WRT54GL. https://www.linksys.com/us/support-product?pid=01t80000003KOkNAAW

[56] Luo, T., Tan, H.-P., Quan, P., Law, Y. W., and Jin, J. Enhancing Responsiveness and Scalability for OpenFlow Networks via Control-Message Quenching. In *Proceedings of International Conference on ICT Convergence (ICTC)* (2012), pp. 348–353.

[57] MaKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: Enabling Innovation in Campus Networks. *Proceedings of ACM SIGCOMM Computer Communication Review 38* (2008), 69–74.

[58] Mininet: An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/.

[59] Network Functions Virtualization (NFV). https://www.etsi.org/committee/nfv

[60] Network Management System (NMS).

https://www.cisco.com/c/en/us/support/docs/availability/high-availability/15114-NMS-bestpractice.html

[61] Open Networking Foundataion (ONF). https://www.opennetworking.org/.

[62] "Open Flow description,"

https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/ accessed: 2016-09-16.

[63] "Lldp," https://www.juniper.net/documentation/us/en/software/junos/multicast-l2/topics/concept/layer-2-services-lldp-overview.html, accessed: 2016-09-21.

[64] L. Ochoa Aday, C. Cervell Pastor, and A. Fernndez, "Current trends of topology discovery in openflow-based software defined networks," 2015. https://upcommons.upc.edu/bitstream/handle/2117/77672/Current+Trends+of+Discovery+Topology+in+SDN.pdf?sequence=1

[65] "OFDP openflow,"

https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf accessed: 2016-09-21.

[66] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," Queue, vol. 11, no. 12, pp. 20:20-20:40, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2559899.2560327

[67] R. Bifulco, J. Boite, M. Bouet, and F. Schneider, "Improving sdn with inspired switches," in Proceedings of the Symposium on SDN Research, ser. SOSR '16. New

York, NY, USA: ACM, 2016, pp. 11:1-11:12. [Online]. Available: http://doi.acm.org/10.1145/2890955.2890962

[68] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3-3. [Online].

[69] HyperFlow: a distributed control plane for OpenFlow Available: http://dl.acm.org/citation.cfm?id=1863133.1863136

[70] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," 2008.

https://cseweb.ucsd.edu//~vahdat/papers/sigcomm08.pdf

[71] "ODL opendaylight," http://www.opendaylight.org/, accessed: 2016-09- 22.

[72] ONOS: towards an open, distributed SDN OS

Available: http://doi.acm.org/10.1145/2620728.2620744

[73] "ONOS projects," http://onosproject.org/, accessed: 2016-09-22.

[74] "Floodlight,"

https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview

accessed: 2016-09-16.

[75] Pox wiki http://intronetworks.cs.luc.edu/auxiliary_files/mininet/poxwiki.pdf

[76] Ryu sdn framework https://ryu-sdn.org/.

[77] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar,E. Salvadori, and B. Snow. Openvirtex: Make your virtual sdns programmable. In Proceedings of the third workshop on Hot topics in software defined networking, pages 25–30. ACM, 2014.

[78] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, and G. Parulkar. Openvirtex: A network hypervisor. In Open Networking Summit 2014 (ONS 2014), 2014. https://openvirtex.com/wp-content/uploads/2014/04/ovx-ons14.pdf

[79] M. Caesar and J. Rexford. Building bug-tolerant routers with virtualization. In Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow, pages 51–56. ACM, 2008.

[80] B. Chandrasekaran and T. Benson. Tolerating sdn application failures with legosdn. In Proceedings of the 13th ACM Workshop on Hot Topics in Networks, page 22. ACM, 2014.

[81] X. Jin, J. Gossels, J. Rexford, and D. Walker. Covisor: A compositional hypervisor for software-defined networks. In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pages 87–101, 2015.

[82] E. Keller, M. Yu, M. Caesar, and J. Rexford. Virtually eliminating router bugs. In Proceedings of the 5th international conference on Emerging networking experiments and technologies, pages 13–24. ACM, 2009.

[83] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou. Feature-based comparison and selection of software defined networking (sdn) controllers.

[84] M. Team. Mininet: An instant virtual network on your laptop (or other pc), 2012.

[85] Campus Network Outage Incident Report for UM IT Systems. https://status.missouri.edu/incidents/r6pkcc5n4ls2.

[86] Amazon Web Services (AWS) Outage - Blackboard Products Affected. https://blackboard.secure.force.com/btbbexportarticlepdf?id=kAA390000004CeGGAU&pdf=true, 2017. Accessed: 2017- 10.

VITA

Haihong Zhu graduated from Shanghai JiaoTong University in 1993 with Bachelor of Science in Electrical Engineering. He received his first Master of Science degree from University of Virginia in Electrical Engineering in 2001. He has worked in the Information Technology Industry for more than 20 years. While working at Cisco, he earned his second Master of Science degree in Software Management in 2016 from Carnegie Mellon University. In Fall 2016, he joined interdisciplinary Ph.D. program at University of Missouri - Kansas City.

His main discipline was Computer Networking and Communication Systems and co-discipline is Electrical and Computer Engineering. His area of research include reliable network, campus network traffic and performance analysis and modeling, software-defined network, wireless sensors, Internet of Things(IoT), Big data, and Artificial Intelligence (AI).