ANALYZING PERFORMANCE OF OPENSTATE IN SOFTWARE DEFINED NETWORK WITH MULTIPLE FAILURES SCENARIOS

Babangida Isyaku

UNIVERSITI TEKNOLOGI MALAYSIA

ANALYZING PERFORMANCE OF OPENSTATE IN SOFTWARE DEFINED NETWORK WITH MULTIPLE FAILURES SCENARIOS

BABANGIDA ISYAKU

A dissertation submitted in partial fulfillment of the
requirements for the award of the degree of
Master of Science (Computer Science)

Faculty of Computing
Universiti Teknologi Malaysia

JANUARY, 2017

I dedicated this research work to my late Father Alh Isyaku Alhassan May Almighty Allah make Jannatul Firdausi to be his final Abode and to my love Mother.

ACKNOWLEDGEMENT

I first thank my Creator, Cherisher and Sustainer for His countless blessings on me. Among the blessings is enabling me to successfully complete this dissertation and the master programme in general.

I like to express my sincere appreciation to entire Management of Faculty of Computing (FC) and Universiti Teknologi Malaysia (UTM) community for providing me with all facilities and resources throughout the research work. Without their supports these journeys would have been difficult one.

Secondly, I also express my sincere gratitude to my thesis supervisor, Assoc. Prof. Dr. Mohd Soperi Mohd Zahid for his immense support and guidance throughout the research work. Our weekly meetings played an important role in my continuous progress and helped me to structure my work schedule.

It is with great pleasure that I take this opportunity to also thank my fellow classmates. Musa wakil Bara, Isah Sani Birnin Gwari and Ibrahim Jafar for the group activities we shared together gave me moments of tranquillity and helped refocus back on my study.

I couldn't possibly forget to express my appreciation to the management of Sule Lamido University, kafin Hausa Jigawa State Nigeria. Especially to the Vice chancellor (Prof. Abdullahi Yusuf Ribadu) and DVC (Prof. Lawan sani Taura) for all that they have done for me, May Almighty Allah reward them abundantly.

Finally, I deeply thank my parents, my brothers for their endless support and unwavering love and prayer. Whenever I felt dim and tired, I could always count on their heart-warming encouragements. May Allah protect and guide them.

ABSTRACT

Software Defined Network (SDN) is an emerging network that decouples the control plane and data planes. Like other networks, SDN undergoes a recovery process upon occurrences of link or node failures. Openflow is considered as the popular standard used in SDN. In Openflow, the process of detecting the failure and communications with controller to recompute alternative path result to long recovery time. However, there is limit with regards time taken to recover from the failures. If it takes more than 50 msec, a lot of packet will be lost, and communication overhead and Round Trip Time (RTT) between switch – controller may be high. Openstate is an Openflow extension that allows a programmer to specify how forwarding rules should be adapted in a stateful fashion. Openstate has been tested only on single failure. This research conduct experiment based on Openstate pipeline design that provides detections mechanism based on switches periodic link probing and fast reroute of traffic flow even when controller is not reachable. In this research, the experiments use Mininet simulation software to analyse and evaluate the performance of Openstate with multiple failure scenarios. The research has compared Overhead communication, Round Trip Time (RTT) between switch – controller and number of packet loss with Openflow and Openstate. On the average, in Openstate packet loss is zero when the recovery time is less than or equal to 70 msec while communication overhead involves 60 packet-in. In Openflow, packet loss is zero when the recovery time is less than or equal to 85 msec while communication overhead involves 100 packet-in. Finally, the average RTTs for Openstate and Openflow are 65 msec and 90 msec respectively. Based on the results obtained, it can be concluded that Openstate has better performance compare to Openflow.

.

ABSTRAK

‘Software Defined Network’ (SDN) adalah satu rangkaian baru yang memisahkan satah kawalan dan satah data. Seperti rangkaian lain, SDN menjalani proses pemulihan selepas hubungan antara dua nod terputus atau nod tidak berfungsi, Openflow adalah suatu piawaian yang digunakan dalam SDN. Dalam proses pemulihan, Openflow mengesan kegagalan dan menyampaikan maklumat itu kepada pengawal untuk mengira hasil jalan alternatif dan boleh menyebabkan masa pemulihan yang panjang. Walau bagaimanapun, terdapat batasan bagi masa yang diambil untuk pulih daripada kegagalan. Jika ia mengambil masa lebih daripada 50 milisaat, banyak paket akan hilang serta beban komunikasi dan masa pergi balik (Round trip time - RTT) antara suis dan pengawal akan menjadi tinggi. Openstate adalah lanjutan Openflow yang membolehkan pengaturcara untuk menentukan bagaimana peraturan penyampaian perlu disesuaikan dengan cara yang dilengkapi keadaan (Stateful). Openstate telah diuji hanya untuk satu kegagalan. Kajian ini menjalankan eksperimen berdasarkan reka bentuk talian paip Openstate yang menyediakan mekanisme pengesanan berdasarkan penyelesaian suis hubungan berkala dan pertukaran laluan pantas aliran trafik walaupun pengawal tidak dapat dihubungi. Eksperimen-eksperimen dibuat menggunakan perisian Mininet untuk menganalisis dan menilai prestasi Openstate dalam senario kegagalan berbilang. Kajian ini meneliti beban komunikasi, RTT antara suis-pengawal dan bilangan kehilangan paket bagi Openflow dan Openstate. Secara purata, kehilangan paket Openstate adalah sifar apabila masa pemulihan adalah kurang atau sama dengan 70 milisaat dan beban komunikasi melibatkan 60 “packet-in”. Bagi Openflow, kehilangan paket adalah sifar apabila masa pemulihan adalah 85 milisaat atau kurang dan beban komunikasi melibatkan 100 “packet-in”. Begitu juga, purata RTT untuk Openstate dan Openflow masing – masing adalah 65 milisaat dan 90 milisaat.

.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACK	-	Acknowledgement
API	-	Application Programming Interface
ARP	-	Address resolution protocol
BFD	-	Bidirectional forwarding detection
CPU	-	Central Processing Unit
ETH	-	Ethernet
FLI	-	File Location identification
FNSS	-	Fast Network Simulation setup
FSM	-	Finite State Machine
HB	-	Heartbeat
ID	-	Identification
IDE	-	Integrated Development tool
IP	-	Internet protocol
MAC	-	Medium Access Control
MB	-	Megabyte
MCA	-	Monitoring Cycle Algorithm
MILP	-	Mixed integer linear programming
MPLS	-	Multiprotocol Label Switching
MS	-	Millisecond
NA	-	Not available
OF	-	Openflow
OFPT	-	Openflow packet
ONF	-	Open Network Foundation
OS	-	Operating system
OVS	-	Open V switch

PKT	-	Packet
RAM	-	Random Access Memory
REST	-	Representation State transfer
RTT	-	Round Trip Time
RYU	-	means flow in germany
SDN	-	Software Defined Networking
SSH	-	Secure Shell
SYN	-	Synchronization
TCAM	-	Ternary Content Addressable Memory
TCP	-	Transport Control Protocol
UDP	-	User datagram Protocol
VM	-	Virtual Machine
XML	-	eXtensible Markup Language

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Overview

Software defined networking is a new paradigm that emerge to offer simplicity over a network through the decoupling of control plane from the underlying forwarding plane (data plane) (Lee Li *et al.* 2014). It offers a single entity called a controller to have a centralized abstract view of the network. Moreover, it creates flexible and dynamic architecture that provide simple network manageability and reliability.

Openflow is largely the most adopted abstraction for the data plane with its match action rules in flow tables (McKeown *et al.,* 2008). Current Openflow abstraction presents some fundamental drawbacks that can prevent an efficient and performing implementation of traffic rerouting schemes. As a matter of fact, in Openflow adaptation and reconfiguration of forwarding rules (i.e. entries in the flow tables) in the data plane pipeline can only be performed by the remote controller, posing limitations on the granularity of the desired monitoring and traffic control due to the overhead and latency required.

Therefore, due to the ineffectiveness of the Openflow to include effective mechanism for fast failure recovery, several efforts from research community to

extend the Openflow specification 1.3 to OpenState have been made in the recent years (A Capone & Cascone, 2014). An Openstate is an extension of Openflow 1.3 that have more additional features which enable the remote controller to enforce control logic to forwarding plane (switches). Openstate protocol has been tested on single failure but is yet to be tested on multiple failure scenarios.

1.2 Problem Background

Software defined networking (SDN) is considered as vital technology for the years to come, Lee *et al.* (2014a) consider it as next future generation network. The central controller is an important entity of SDN. It performs functionality such as monitoring, modification and computation of the forwarding rules. Moreover, it allows the flexibility to directly configure the infrastructure devices (data planes) (Adrichem *et al.* 2014a).

There is no doubt that the controller offers great advantage to the network, but there are some overhead that will be involved to restore the network back to operational state, after the occurrences of failures. Failures may occur due to several events, such as software or hardware failure, and node or link could be down due to fibre cut or interface break. In the earlier version of Openflow 1.0 it supports only one single flow table and secure channel to controller. When a failure occurs, the switch that detect the failure send notification to the controller through `packet in` message. The controller then locates the point of the failure and computes new suitable path and update the flow table of the affected switch with the new path. Therefore, Openflow 1.1 onward emerge with additional features which support multiple table, incorporated with fast failover group table functionality to speed up local failure detection and recovery without imposing much extra processing load on controller. The fast failover group table extend Openflow configuration rules that allows monitoring and

forwarding of packet at switch label. The group table is pre-configured to monitor the status of switch port. The table has several action buckets populated with different forwarding action. Therefore, when failure occurs, the switch that detect the failures perform lookup in the table and switchover to alternative path. In case of no alternative path found on the node that detect the failure, crank bank signalling is performed to keep rerouting the packet to neighbouring node until alternative path is found. Obviously, with the fast failover, controller will be relieved with some extra processing load but the recovery time may increase due to the rerouting of packet using the crankbank signalling. Secondly with the Openflow fast failover detect node and reroute node are always the same. Unfortunately, such a solution is not always feasible, as it strongly depends on topology and capacity constraints.

In Openstate, the fault detection event is not immediately communicated to controller but rather the switch that detect the failure tag the data packet and forward it back to a node called the reroute node. The reroute node will then execute state transition and find a suitable new path that can be used to deliver future data packets and inform the controller about the topology changes.

Therefore, the main activities to recover the network after failure include the detection time and restoration time. The restoration time includes the propagation time to notify the central controller about the event, path re-computation and reconfiguration of the network by the central controller (Adrichem *et al.* 2014b, Sharma & Staessens, 2013). Adrichem *et al.* (2014c) further emphasize in Openflow 1.0 network, the time taken for the controller to initiate path restoration is over 100msec excluding detection time. According to another author Lee *et al*. (2014b) it takes from 260 msec to 310 msec for Openflow controller to set up the recovery path after failure detection. This range of time is considered too long as the acceptable time required by provider network is at most 50 msec (Adrichem *et al*., 2014d). Hence, several efforts have been made recently to reduce the restoration time of Openflow controller.

Goransson & Black (2014), Asten, (2014) has identified that the long restoration time may due to the computational load on the controller that is too much. Furthermore, Lee *et al*. (2014) emphasize that bottleneck at the controller increases as the network size grows larger. Thus, there is need for an appropriate mechanism to minimize the load on the Openflow controller.

Openstate is an extended version of Openflow 1.3. It is introduced to minimise the computational load on the controller and have the facility for quick failure recovery. In Openflow 1.0 when a failure occurs, the controller must recompute the new path but in Openflow 1.1 onward crankback signalling can be performed without immediately communication to controller whereas in Openstate data packet are tagged and bounced it back to the reroute node to enable detour. Thus, the Openstate network has less load than the Openflow network. Openstate network also promote quicker recovery time than Openflow network because the new path can quickly be enabled by the switch without consulting the controller. The delivery of future data packets using the new path can be activated without waiting for the instruction from the controller. In summary, the advantages of Openstate compared to Openflow are described in Table 1.1.

Table 1.1: Comparison between Openflow and Openstate

Protocol	**Controller Computational Load**	**Failure Recovery**
Openflow	Fast failover: local reroute based on port state, switch to detour. In case no local reroute. Controller Compute and Configure new path	Wait for controller to compute new path. Controller is involved in recovery process. Time range from 100 msec to 360 msec

Openstate	Computation of backup path are precomputed	Switch activate the new path. Time close to 50 msec

To the best of our knowledge, the Openstate has been evaluated only on single failure scenarios by A Capone & Cascone, (2014). The occurrences of multiple and simultaneous failures may happen and sometimes cannot be avoided, or beyond control (Steven *et.al*. 2014). Thus, in this research, the study would like to evaluate and analyse the performance of Openstate on Multiple failure scenarios.

It is expected that the overhead of Openstate controller will increase in multiple failure scenarios. For example, when n failures occur, the first failure can be resolved without controller intervention, but in the subsequent failures, the detect switches will be busy performing lookup to find match for alternative path in the flow table. In case no local backup path is available, the detect nodes switchover and send number of `packet-in` to controller for reactive support. This study plan to evaluate the performance with the implementation correspond to reactive Openflow network. Since Openflow 1.1 onward uses fast failover group that perform crankback signalling for fast failure recovery and this happen to be similar with the approach for multiple failure in Openstate. For both protocols the study will consider the communication overhead, packet loss and Round trip time for the flow affected by fault.

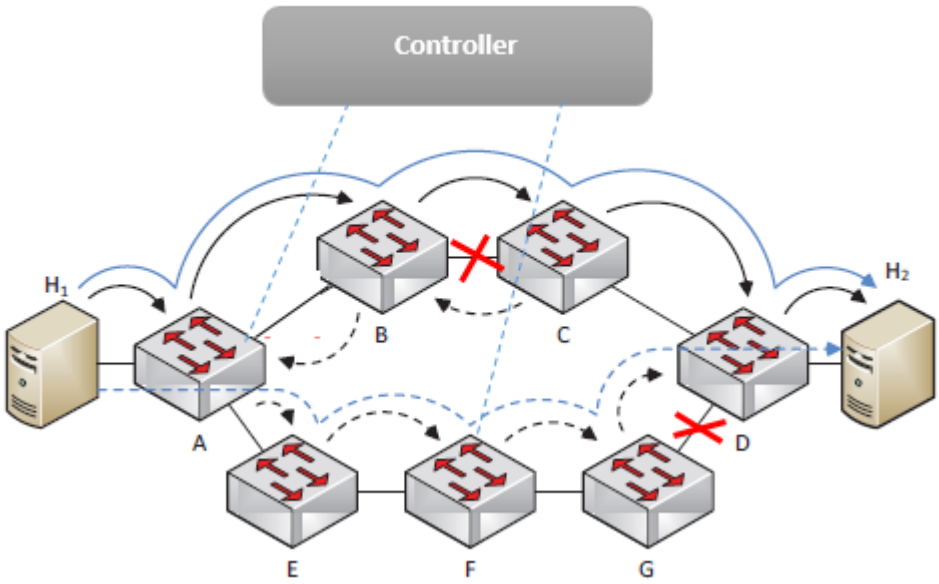

Figure 1.1 Network topology with multiple failure

For example, in Figure 1, H1 wants to send data packet to host H2, all data packet is routed through optimal primary path (A →B→C→D) on normal condition. However, introducing link failure between switch B →C and G → D. The fault event is noticeable by detect switch: B and G. Switch A and switch F are reroute nodes. If both switch B and switch G have no local back-up paths available in state table, then both will be busy sending `packet-in` to controller to seek for reactive support. Assume that switch B and switch G communicate with the controller at time t_1 and time t_2. Thus, the controller will perform the new back-up paths computation two times at the same time or one at slightly after t_1 and another one slightly after t_2,. This will result to more round trip time (RTT) between the detect switch and controller. However, this study want to analyse the performance of Openstate considering multiple failures, using the following approach, for the first failure to be resolve without controller intervention whereas subsequent one can seek for reactive support of controller using `packet-in` message to controller to compute backup paths for both switches. In the case of subsequent `packet-in` the flow are forwarded without triggering `packet-in`. Obviously, there will be some overhead and Round Trip Time (RTT). The study want to evaluate and analyse the performance of the Openstate through Overhead Communication from involved switch – controller, Round trip time

and finally observed the number of packet loss, the simulation will be performed using testbed based on mininet.

1.3 Problem Statement

Obviously, it is important to reduce the processing load on the controller. This would help to speed up the recovery time of SDN upon occurrences of failures. The more time taken to recover from failures, the more data packets may be losses as it significantly affects the network performance. Currently Openstate allows SDN to converge in close to 50 msec time required by network provider as indicated in A Capone & Cascone, (2014). However, Openstate has been tested on single failure only. It is important to study whether Openstate will also converge in close to 50 msec in case of multiple failures. Recovery of multiple failure involve more processing load on the controller and communication overhead between the controller and involved switch in SDN network. In this research, the study want to analyses the performance of Openstate on multiple failures scenarios through communication overhead, RTT and number of packet loses.

1.4 Goal

The goals of this research is to test multiple failures scenarios in Openstate, to observes the time taking for Openstate to recover from multiple failures, and also to study whether Openstate recover faster than Openflow from failures.

1.5 Objectives

To solve the research problem, this research considers the following objectives.

1. To design failure scenarios for evaluating recovery process of SDN
2. To implement failure recovery scheme of Openflow and Openstate in mininet simulator
3. To compare and evaluate the performance of Openstate and Openflow in SDN with multiple and single failure scenarios.

1.6 Research Scope

To achieve the above listed objectives, the research focuses on the following scope:

- ✓ Mininet network simulator will be used as the simulation tool in this research for simulating network interface with failures
- ✓ Type of network; software defined networking
- ✓ Software defined networking protocol: Openstate and Openflow
- ✓ Analysis was limited to overhead communication switch-controller, number of packet loss and Round Trip Time for the flow affected by fault.
- ✓ The experiment is limited to; Controller (ryu), switch (ofsoftswitch13) and network topology (Norway).

1.7 Significant of the Research

Openstate has been tested on single failure only, multiple failures cannot be avoided. However, Openstate need to be tested on multiple failures scenarios and evaluate how long it will take to recover from failures. The results can be used to decide whether Openstate should be modified to have quicker recovery time. Therefore, we believe this research work can close one gaps by analyzing the performance of Openstate considering multiple failures.

1.8 Summary

To summarize the chapter, this research address failure recovery in software defined networking we present the following: Overview of software define networking, the background of the research, the protocols used in SDN, the study presented failure scenario, how the Openstate and Openflow works and differ from each other, the objective and significant of the research. Finally, the Dissertation outline of this research work is organized as follows.

Chapter 2 give a general view of SDN architecture. Discusses the adoption and reconfiguration of SDN standard protocol, i.e. Openflow and Openstate with some of the related work in failure detection and recovery. Platform resources used in simulating failure scenarios in (SDN).

Chapter 3 discusses the overview of the research methodology along with the framework for the study and the overall research plan.

Chapter 4 provides a detailed explanation on how the study design and implement the experiment simulation for failure scenarios using testbed based on mininet simulator. Based on Openstate considering multiple failure scenarios.

Chapter 5 present the experimental results. The chapter make comparison between Openflow and Openstate. Finally, discussion of the experimental findings will be given

Chapter 6 concludes the study and provides a direction for future works.

REFERENCES

Adrichem, N. L. M. Van, Asten, B. J. Van, & Kuipers, F. a. (2014). Fast Recovery in Software-Defined Networks. *2014 Third European Workshop on Software Defined Networks*, 61–66.

Ahuja, S. S., Ramasubramanian, S., & Krunz, M. M. (2009). Single-link failure detection in all-optical networks using monitoring cycles and paths. *IEEE/ACM Transactions on Networking*, *17*(4), 1080–1093.

Al-somaidai, M. B., & Yahya, E. B. (2015). Effects of Linux Scheduling Algorithms on Mininet Network Performance, *3*(5), 128–136.

Apostolidis, P. (2016). Network management aspects in SDN, (December

Arshad, N., Heimbigner, D., & Wolf, A. L. (2004). A planning based approach to failure recovery in distributed systems. *WOSS'04: 1st ACM SIGSOFT Workshop on Self-Managed Systems*, 8–12.

Asten, B. van. (2014). Scalability and Resilience of Software-Defined Networking: An Overview. *arXiv Preprint arXiv* 1–19. Retrieved from http://arxiv.org/abs/1408.6760

Atary, A. (n.d.). Efficient Round-Trip Time Monitoring in OpenFlow Networks.

Aweya, J. (2001). IP router architectures: An overview. *International Journal of Communication Systems*, *14*(5), 447–475.

Azizi, M., Benaini, R., & Mamoun, M. Ben. (2015). Delay Measurement in Openflow-Enabled MPLS-TP Network, *9*(3), 90–101.

Bianchi, G., Bonola, M., & Capone, A. (2014). Towards wire-speed platform-agnostic control of OpenFlow switches. *arXiv Preprint arXiv: …*. Retrieved from http://arxiv.org/abs/1409.0242

Bianchi, G., Bonola, M., Capone, A., & Cascone, C. (2014). OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch. *ACM SIGCOMM Computer Communication Review*, *44*(2), 44–51.

Bianchi, G., Capone, A., Bonola, M., Bianchi, G., & Bonola, M. (2014). Public Review for OpenState : Programming Platform-independent Stateful OpenFlow Applications Inside the Switch a c m s i g c o m m OpenState : Programming Platform-independent Stateful OpenFlow Applications Inside the Switch, *44*(2), 44–51.

Braun, W., & Menth, M. (2014). Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices, 302–336.

Capone, A., & Cascone, C. (2014). Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState. *arXiv Prepr. arXiv {…}*.

Capone, A., Cascone, C., Pollini, L., Sanvito, D., & Milano, P. (2015). P4 Implementation of a Stateful Data Plane and its Application to Failure Recovery Beba, (April 2014), 2015.

Capone, A., & Cascone, C. (2015). Supporting Stateful Forwarding in P4 Stateless dataplane.

Capone, A., Cascone, C., Nguyen, A. Q. T., Pollini, L., Sans, B., & Sanvito, D. (2015). Fast and reliable fault recovery in SDN with OpenState.

Cascone, C., Pollini, L., Sanvito, D., & Capone, A. (n.d.). Traffic Management Applications for Stateful SDN Data Plane.

Chik, J. K., Lindberg, U., & Schutt, C. E. (1996). The structure of an open state of beta-actin at 2.65 A resolution. *Journal of Molecular Biology*, *263*(4), 607–623.

Conejo, A. J., & Series, A. M. (2002). Mixed-Integer Linear Programming.

Degree, M., Engineering, T., Author, M., Jos, D., Marti, Q., Cervell, C., & Date, P. (2015). Master thesis

Demo, O. L. (2015). OpenState demo.

Fallis, A. . (2013). No Title No Title. *Journal of Chemical Information and Modeling*, *53*(9), 1689–1699.

Favaro, A., & Ribeiro, E. P. (n.d.). Reducing SDN / OpenFlow Control Plane Overhead with Blackhole Mechanism.

Ghodsi, A., Shenker, S., Koponen, T., Singla, A., Raghavan, B., & Wilcox, J. (2011). Intelligent design enables architectural evolution. *Proceedings of the 10th ACM Workshop on Hot Topics in Networks - HotNets '11*, 1–6.

Goransson, P., & Black, C. (2014). Software Defined Networks. *Software Defined Networks*, 259–279.

Gordon, G. J., Hong, S. A., & Dudík, M. (2009). First-order mixed integer linear programming. *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 213–222. Retrieved from http://dl.acm.org/citation.cfm?id=1795114.1795140

Hamad, D. J., Yalda, K. G., & Okumus, I. T. (n.d.). Getting traffic statistics from network devices in an SDN environment using OpenFlow, 951–956

Heller, B. (2009). OpenFlow Switch Specification. *Current*, *0*, 1–36.

Heller, B. (2009). OpenFlow Switch Specification. *Current*, *0*, 1–36.

Hudyma, R., & Fels, D. I. (n.d.). Categories of Network Failures.

Jammal, M., Singh, T., Shami, A., Asal, R., & Li, Y. (2014). Software defined networking: State of the art and research challenges. *Computer Networks*, *72*, 74–98.

Jing-Quan Li, P. B. M. and D. B. (2006). The k -Degree Cayley Graph and its Topological Properties. *Networks*, *47*(1), 26–36.

Kamamura, S., Masuda, A., & Sasayama, K. (2012). Autonomous Fast Rerouting for Software Defined Network. *NICT International Workshop,APII 2012*, 1–22.

Kandula, S., Katabi, D., & Berger, A. (2007). Dynamic Load Balancing Without Packet Reordering, *37*(2), 53–62.

Kannan, K., & Banerjee, S. (2013). Compact TCAM : Flow Entry Compaction in TCAM for Power Aware SDN, 439–444.

Kaur, K., & Kaur, S. (2016). Performance Analysis Of Python Based OpenFlow Controllers, 189–192.

Kim, E., Lee, S., Choi, Y., Shin, M., & Kim, H. (n.d.). A Flow Entry Management Scheme for Reducing Controller Overhead.

Kim, S., & Lumetta, S. S. (n.d.). Evaluation of Protection Reconfiguration for Multiple Failures in Optical Networks.

Kreutz, D., Ramos, F. M. V, Verissimo, P., Rothenberg, C. E., Azodolmolky, S., Member, S., & Uhlig, S. (n.d.). Software-Defined Networking : A Comprehensive Survey, 1–61.

Kumar, D., & Sood, M. (2014). Software Defined Networking: A Concept and Related Issues. *International Journal of Advanced Networking & Applications*, *6*(2), 2233–2239. Retrieved from

http://search.ebscohost.com/login.aspx?direct=true&db=iih&AN=99879283&lang=es&site=ehost-live

Kvalbein, A., Hansen, A. F., Gjessing, S., & Lysne, O. (n.d.). Fast IP Network Recovery using Multiple Routing Configurations

Lee, S. S. W., Li, K. Y., Chan, K. Y., Lai, G. H., & Chung, Y. C. (2014). Path layout planning and software based fast failure detection in survivable OpenFlow networks. *DRCN 2014 - Proceedings, 10th International Conference on Design of Reliable Communication Networks*.

Liu, G., Ji, C., & Member, S. (n.d.). Scalability of Network-Failure Resilience, *2*, 1–14.

Malucelli, F. (n.d.). Integer Linear Programming, 1–50.

Martinez, C., Ferro, R., & Ruiz, W. (2015). Next generation networks under the SDN and OpenFlow protocol architecture.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., … Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, *38*(2), 69.

Meador, B. (2008). A Survey of Computer Network Topology and Analysis Examples Keywords :, 1–11. Retrieved from http://www.cs.wustl.edu/~jain/cse567-08/ftp/topology/index.html

Menabo, A. (2015). Evaluating SDN and SDN-based Multicast for Network Intensive Services in UNINETT.

Molina, E., Jacob, E., Matias, J., Moreira, N., & Astarloa, A. (2015). Availability Improvement of Layer 2 Seamless Networks Using OpenFlow, *2015*(v).

Nacshon, L. (n.d.). Floware : Balanced Flow Monitoring in Software Defined Networks.

Narisetty, R., Dane, L., Malishevskiy, A., Gurkan, D., Bailey, S., Narayan, S., & Mysore, S. (2013). OpenFlow configuration protocol: Implementation for the of management plane. *Proceedings - 2013 2nd GENI Research and Educational Experiment Workshop, GREE 2013*, 66–67.

Nokia, V. K. (2016). M . Sc . Thesis : Analysis of OpenFlow Protocol in Local Area Networks, (August 2013).

Open Networking Foundation. (2013). SDN Architecture Overview. *Onf*, (1), 1–5.

Orlowski, S., & Wessäly, R. (2010). SNDlib 1 . 0 — Survivable Network Design Library.

Paris, S., & Paschos, G. S. (n.d.). Dynamic Control for Failure Recovery and Flow Reconfiguration in SDN.

Pontarelli, S., Bonola, M., Bianchi, G., Capone, A., & Cascone, C. (n.d.). Stateful Openflow : Hardware Proof of Concept

Proposition, V. (2015). Segment Routing : Prepare Your Network for New Business Models, 1–16.

Romero, G., & Muntaner, D. T. (2012). Evaluation of OpenFlow Controllers.

Sahri, N. M., & Okamura, K. (2014). Openflow Path Fast Failover Fast Convergence Mechanism, 23–28.

Sessini, P., & Mahanti, A. (n.d.). Observations on Round-Trip Times of TCP Connections.

Sharma, S., & Staessens, D. (2013). Fast failure recovery for in-band OpenFlow networks. *Design of Reliable …*, (1), 52–59. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6529882

Skinner, J. E., & Moln??r, M. (1999). Event-related dimensional reductions in the primary auditory cortex of the conscious cat are revealed by new techniques for enhancing the non-linear dimensional algorithms. *International Journal of Psychophysiology*, *34*(1), 21–35.

Smith, J., & Taskin, Z. (2008). A Tutorial Guide to Mixed Integer Programming Models and Solution Techniques. *Optimization in Medicine and Biology*, 1–23.

Srikanth, K., Kingston, S., & Bhaskar, R. (2011). SDN and Open Flow : A Tutorial.

Survey on Recovery from Failure in Software Define Networking (SDN). (n.d.), 1–8.

Technology, B. F. D., & Paper, W. (n.d.). BFD Technology White Paper, 1–14.

Tilmans, O., Bonaventure, O., Vissicchio, S., & Canini, M. (2014). Robust fault-recovery in Software-Defined Networks.

Tkachova, O., & Saad, I. (2015). Method for OpenFlow Protocol Verification, 139–140.

Turull, D., Hidell, M., & Sj, P. (n.d.). Performance evaluation of OpenFlow controllers for network virtualization.

Wang, S. (n.d.). Comparison of SDN OpenFlow Network Simulator and Emulators : EstiNet vs . Mininet.

Yu, C., Lumezanu, C., Zhang, Y., Singh, V., Jiang, G., & Madhyastha, H. V. (n.d.). FlowSense : Monitoring Network Utilization with Zero Measurement Cost 1 Introduction.

Zhou, Y., Lakamraju, V., Koren, I., Krishna, C. M., & Member, S. (2007). Software-Based Failure Detection and Recovery in Programmable Network Interfaces, *18*(11), 1539–1550.