FAULT TOLERANCE IN THE DATA PLANE OF SOFTWARE-DEFINED NETWORKS

by

Barış Yamansavaşçılar

B.S., Computer Engineering, Yıldız Technical University, 2015

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2019

ACKNOWLEDGEMENTS

If you have a supervisor like Prof. Cem Ersoy, your life as a MS student is enjoyable since you are regularly provided with elegant lunches and dinners, and you are invited to cozy activities during your studies. He is my role model and I am so lucky to have such a supervisor. Moreover, if you have a co-advisor like Assist. Prof. Atay Özgövde, the details of your work can create new paths and therefore inspire novel papers throughout your studies. Furthermore, if you have a deus ex machina in your lab like Ahmet Cihat Baktır, you can survive the most desperate situations when you are stuck in a problem. I am very appreciated that I worked with these beautiful people and benefited their wisdom during my MS studies.

I would also like to express my thankfulness to Prof. Tuna Tuğcu, since he opened new horizons to me by involving me into different kind of projects, especially the Colibri Project. By involving in them, I was able to improve my abilities in group projects and presentations.

I would also like to thank my colleagues in NETLAB including Can Tunca, Orhan Ermiş, Raşit Mete Eşrefoğlu, Hüseyin Anıl Özmen, Gökcan Çantalı, Niaz Chalabianloo, Serhan Daniş, Meriç Turan, and Alper Alimoğlu for their friendship, advice, and helpfulness. Without their companionship, I would not have such a great experience during my graduate years.

Finally, I would like to special thank to my sister, Mısra Yamansavaşçılar, and my mother, Nuray Özyavaş whom I dedicate this thesis. Without their support, I would never achieve my accomplishments.

ABSTRACT

FAULT TOLERANCE IN THE DATA PLANE OF SOFTWARE-DEFINED NETWORKS

Failures in networks result in service disruptions which may cause deteriorated Quality of Service for the end users. Since SDN is becoming the mainstream paradigm for networks, especially for data centers, correct implementation of the fault tolerance for SDN-based networks is crucial. Current SDN data plane fault tolerance approaches can be classified as reactive and proactive which may or may not rely on the involvement of the central controller, respectively. However, none of them qualify as complete solutions, providing only partial remedies. In this work, we propose a dynamic protection approach that considers not only the existing faults within the network but also the quality of alternative paths. We also investigate how application based parameters are affected by possible failures. To this end, we explore the change in Quality of Experience (QoE) caused by link failures under different cases using Dynamic Adaptive Streaming over HTTP (DASH) for video streaming.

Since tremendous amount of traffic is generated in modern networks, the congestion issue is faced more frequently. Even though the video streaming paradigm, DASH, is proposed as a solution for the changing condition of the networks, it is not sufficient considering the heavily loaded links that show the symptoms of the link failures. Therefore, the flexible implementation of the data plane fault tolerance scheme that can be applied to other problems like congestion in networks is crucial. Thus, we apply the data plane fault tolerance approach in SDN to improve the QoE of DASH clients in the case of congestion as well as the failure.

ÖZET

YAZILIM TANIMLI AĞLARDAKİ VERİ DÜZLEMİNDE ARIZA DAYANIKLILIĞI

Ağlardaki arızalar, son kullanıcılar için hizmet kalitesinin kötüleşmesine sebep olabilen hizmet kesintilerine yol açmaktadır. Yazılım Tanımlı Ağlar (YTA), özellikle de veri merkezleri için, ana akım olmaya başladığından, YTA tabanlı ağlar için arıza toleransının doğru şekilde uygulanması çok önemlidir. Mevcut YTA veri düzlemi arıza toleransı yaklaşımları, sırasıyla merkezi kontrol cihazının katılımına dayanan veya dayanmayan reaktif ve proaktif olarak sınıflandırılabilir. Ancak, bu yaklaşımlar arızalar için kısmi çözümler sunmakta olup, bütünsel bir çözüm sunamamaktadır. Bu çalışmada, sadece ağdaki mevcut hataları değil, alternatif yolların kalitesini de dikkate alan dinamik (proaktif) bir koruma yaklaşımı öneriyoruz. Ayrıca, uygulama tabanlı parametrelerin olası arızalardan nasıl etkilendiğini de incelemekteyiz. Bu amaçla, video akışı için HTTP üzerinden Dinamik Uyarlamalı Akış (HDUA) kullanarak farklı durumlardaki bağlantı arızalarından kaynaklanan Deneyim Kalitesindeki (QoE) değişimi araştırıyoruz.

Modern ağlarda muazzam miktarda trafik oluştuğu için tıkanıklık sorunu daha sık yaşanmaktadır. Video akışı paradigması, HDUA, ağların değişen koşullarına bir çözüm olarak önerilse de, bağlantı arıza semptomlarını gösteren yoğun yüklü bağlantılar göz önüne alındığında yeterli değildir. Bu nedenle, ağlardaki tıkanıklık gibi diğer sorunlara da uygulanabilecek esnek bir veri düzlemi arıza toleransı mekanizmasının gerçekleştirilmesi çok önemlidir. Bu nedenle, bu çalışmada, hem arıza hem de tıkanma durumlarında HDUA istemcilerinin deneyim kalitesini geliştirmek için YTA'daki veri düzlemi arıza toleransı yaklaşımını uyguluyoruz.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . viii
LIST OF TABLES . . . xii
LIST OF SYMBOLS . . . xiii
LIST OF ACRONYMS/ABBREVIATIONS . . . xiv
1. INTRODUCTION . . . 1
2. BACKGROUND . . . 6
2.1. Software-Defined Networking . . . 6
2.2. Fault Tolerance in SDN Data Plane . . . 7
2.2.1. Failure Detection . . . 8
2.2.1.1. Loss of Signal (LoS) . . . 8
2.2.1.2. Link Layer Discovery Protocol (LLDP) . . . 8
2.2.1.3. Bidirectional Forwarding Detection (BFD) . . . 8
2.2.2. Path Calculation . . . 9
2.2.2.1. Restoration . . . 9
2.2.2.2. Protection . . . 10
2.2.3. OpenFlow Groups . . . 10
2.3. Video Streaming and Quality of Experience . . . 12
3. LITERATURE REVIEW . . . 16
3.1. Restoration Approaches . . . 16
3.2. Protection Approaches . . . 17
3.3. DASH and QoE in SDN . . . 18
3.4. Congestion in SDN . . . 19
4. DESIGN OF THE FAULT TOLERANT DATA PLANE . . . 20
4.1. Restoration Module . . . 20
4.2. Static Protection Module . . . 21
4.3. Dynamic Protection Module . . . 23

4.4. BFD-based Congestion Detection Module 25
4.4.1. Congestion Detection 26
4.4.2. Rerouting Flows 27
4.5. DASH Module 27
5. PERFORMANCE EVALUATION OF NETWORK-BASED METRICS . . . 30
5.1. Recovery Time vs Packet Loss 31
5.2. Effect of the Number of Flows 31
5.3. Effect of the Size of Topologies 38
5.4. Evaluation of Linux Bridge and BFD Intervals 38
5.5. Evaluation of the Quality of the Alternative Paths 44
6. PERFORMANCE EVALUATION OF APPLICATION-BASED METRICS . 47
6.1. Restoration 47
6.2. Static Protection 48
6.3. Dynamic Protection 48
7. PERFORMANCE EVALUATION OF BFD-BASED CONGESTION DETECTION ON QoE 52
7.1. Effect of Segment Size 53
7.2. The Effect of Traffic Load 56
7.3. The Effect of BFD Intervals 56
8. CONCLUSIONS 60
REFERENCES 62

LIST OF FIGURES

Figure 2.1. Traditional networking vs SDN 6

Figure 2.2. Restoration in SDN. 9

Figure 2.3. OpenFlow Groups. 11

Figure 2.4. The working mechanism of a Fast Failover group in OpenFlow switch. 12

Figure 2.5. Concept of DASH 14

Figure 2.6. Relation between video quality and bitrate for three different resolutions. 15

Figure 4.1. The Algorithm of Restoration Module 21

Figure 4.2. Static Protection in SDN. 22

Figure 4.3. The concept of the dynamic protection module. 24

Figure 4.4. The design of the BFD-based Congestion Detection System. . . . 26

Figure 4.5. The Algorithm of Static Protection Module 28

Figure 4.6. The Algorithm of Dynamic Protection Module 29

Figure 5.1. The topology using a Linux bridge. 32

Figure 5.2. The topology without using a Linux Bridge. 32

Figure 5.3. Comparison of packet loss and recovery time including restoration and static protection modules. 33

Figure 5.4. The packet loss rate based on the flow count and used fault tolerance method. 35

Figure 5.5. The number of packets of the control path traffic during the lifetime of the experiment. 36

Figure 5.6. The number of packets of the control path traffic during the lifetime of the experiment. 37

Figure 5.7. *TwoPathsTopology* that has two different alternative paths after the failure at the link between Switch 2 and 5. 39

Figure 5.8. *FivePathsTopology* that has five different alternative paths after the failure at the link between Switch 2 and 5. 39

Figure 5.9. *FourteenPathsTopology* that has 14 different alternative paths after the failure at the link between Switch 2 and 5. 40

Figure 5.10. *TwentyNinePathsTopology* that has 29 different alternative paths after the failure at the link between Switch 2 and 5. 40

Figure 5.11. The packet loss based on the different topologies. 41

Figure 5.12. The demonstrative examples of failure recovery for different scenarios. 42

Figure 5.13. BFD significantly reduces recovery time independent from failure methods. 43

Figure 5.14. The transmission of packets stop when we use Linux bridge for the failure for static protection module. 44

Figure 5.15. Packet Loss based on the BFD message interval. 44

Figure 5.16. Comparison of the QoAP and traditional protection approach based on packet loss for each QoAP interval. 46

Figure 5.17. Differences between static and dynamic protection modules considering quality of alternative paths. 46

Figure 6.1. The effect of failure recovery on the video quality and buffer level of video streaming using restoration module with Mininet-based failure. 49

Figure 6.2. The effect of failure recovery on the video quality and buffer level of video streaming using restoration module with Linux bridge-based failure. 50

Figure 6.3. QoE is significantly affected if the quality of the alternative paths is not considered for fault tolerance. 51

Figure 7.1. The topology and video traffic route 53

Figure 7.2. The change of QoE parameters for the congested link with 98% load. 54

Figure 7.3. The change of QoE parameters for the congested link with 98% load. 55

Figure 7.4. The impact of the traffic load on the average video quality with respect to different segment sizes 57

Figure 7.5. The impact of the traffic load on the quality switch count with respect to different segment sizes. 58

LIST OF TABLES

Table 2.1. Bitrates for different screen resolutions 15

Table 2.2. Coefficients of the generalized video quality function 15

LIST OF SYMBOLS

M	Detection Time Multiplier
T_d	Failure Detection Time
T_i	Message Transmit Interval
T_p	Propagation Time
T_{pc}	Path Calculation Time
T_{qoap}	Quality of Alternative Path Calculation Interval
T_r	Failure Recovery Time
T_{update}	Switch Update Time

LIST OF ACRONYMS/ABBREVIATIONS

API	Application Programming Interface
BFD	Bidirectional Forwarding Detection
BFS	Breadth First Search
BGP	Border Gateway Protocol
DASH	Dynamic Adaptive Streaming over HTTP
DFS	Depth First Search
FF	Fast Failover
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LAN	Local Area Network
LFM	Link Failure Messages
LLDP	Link Layer Discovery Protocol
LoS	Loss of Signal
MEC	Mobile Edge Computing
MOS	Mean Opinion Score
MPD	Media Presentation Description
OSPF	Open Shortest Path First
QoAP	Quality of Alternative Path
QoE	Quality of Experience
QoS	Quality of Service
RTSP	Real-Time Streaming Protocol
SDN	Software-Defined Networking
SLA	Service Level Agreement
SSIM	Structural Similarity Index
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network

VoIP	Voice over IP
WAN	Wide Area Network

1. INTRODUCTION

Alleviating failures is crucial for service providers for designing robust systems in order to meet the expected Quality of Service (QoS) of their clients' Service Level Agreement (SLA) [1]. Strict SLA requirements when combined with the risk of reputation loss, are evident that network and service providers should consider fault handling within their system as critical. Ideally, network failures should be handled seamlessly, transparent to the end user, not affecting their Quality of Experience (QoE). Practically, however, remedies try to achieve minimizing the effect of the faults and avoid disruptions in the network level QoS and application level QoE as much as possible.

A fault once occurred in a network evidently impacts its operation. Therefore, we broadly categorize this effect as the impact on QoS behavior and the impact on QoE behavior of the network. Here, the QoS parameters represent the application independent network characteristics whereas QoE parameters represent the specific impact that an end user experiences for a given application. Both QoS and QoE parameters are of importance for assessing how a given fault triggers fluctuations in the network performance.

Currently, video streaming is the most influential traffic type of the Internet since it occupies 73% of the overall data traffic volume [2]. Moreover, it is expected to occupy 82% of all consumer Internet traffic by 2021 according to the forecast report by Cisco [3]. Apart from its major role in the traffic composition, video streaming also forms a separate category with respect to other applications since it is continuously evaluated by the end user. This evaluation reflects itself as QoE of the user which is essential for the video service providers since low QoE may result in financial as well as reputation loss for them.

Currently, Dynamic Adaptive Streaming over HTTP (DASH) is the widely accepted technology for video streaming. DASH, in accordance with the stateless character of HTTP, can receive video segments in independent connections, which in turn

allows for versatile and dynamic streaming operation. This is in contrast with traditional streaming protocols such as Real-Time Streaming Protocol (RTSP) that work in a stateful manner by tracking the state of the clients during the streaming. Thus, DASH features render video streaming to be carried out in a much more dynamic way so that it provides many advantages including the adaptation to the network conditions [4, 5].

On the other hand, assuming a continuum of network performance degradation, link failure can be considered as an extreme case of congestion in which the delay becomes infinity. Bearing this view in mind, fault tolerance solutions originally developed for link failures can, in fact, serve as candidate methods to deal with congestion when carefully adjusted to this new context. Accordingly, even though the adaptive nature of DASH provides important flexibility considering the unstable network conditions, its capabilities are insufficient to handle the extreme congestion case as well as the link failure. Due to the temporary inadequate link capacity, adaptive bitrate and dynamic buffering features of the DASH client may not be sufficient to leverage the deteriorating QoE after a serious traffic load.

Software-Defined Networking (SDN) on the other hand, provides opportunities in terms of fault tolerance with its central view on the whole network. In SDN, the concept of fault tolerance can be taken into consideration within three domains: the data plane, the control plane, and the application plane. The data plane issues consist of link or switch failures whereas the control plane domain considers the failure of the switch-controller connections or the failure of the controller itself. The application plane domain focuses on the failure of an application that can affect the northbound API which in turn can cascadingly affect other applications. In this thesis, we focus on the fault tolerance within the data plane in SDN.

In SDN, there are two essential approaches that can be used for the data plane fault tolerance problem: (i) restoration (reactive approach) and (ii) protection (proactive approach). Both of these approaches have their respective advantages and disadvantages in terms of the recovery time and recent network view [6, 7]. In restoration,

the new rules for the affected flows are computed considering the recent network view, while in protection, the new rules are predefined in case of a failure. Protection is usually accepted as a superior approach since it reduces the recovery time significantly compared to the restoration. However, this view does not correctly reflect all the requirements of a fault protection mechanism since it omits the quality of the new path that is used to substitute the faulty one. Typically, in a network, various alternative paths exist to recover a given fault and the quality of these paths should be incorporated into the selection process. Thus, we propose Quality of Alternative Path (QoAP) based dynamic protection which combines both the restoration and the protection methods to achieve fast recovery while selecting among available high-quality paths.

Although it seems promising to mitigate failures and congestions using fault tolerance methods, sensitivity of the mechanisms for detecting a data plane fault is also crucial. There are three essential failure detection methods used in networks including Loss of Signal (LoS), Link Layer Discovery Protocol (LLDP), and Bidirectional Forwarding Detection (BFD). Among these methods, the BFD protocol has superiority since it is designed only for failure detection and its detection speed outperforms the others [8, 9]. Thus, its use in a fault tolerant system provides many benefits for both end users considering QoE and service providers to meet the expected service quality.

Another focus of our work is related to the experimentation method that is typically being used in the literature. Mininet, which owes its reputation to the wide variety of capabilities that it offers, is the emulation environment for evaluating SDN related proposals in a research laboratory setting [10]. However, in the context of producing link failures, we argue that commands provided by Mininet are insufficient to correctly emulate a fault scenario since it actually destroys the whole connection including the ports of switches. Moreover, Mininet allows switches to notify port failures to the northbound applications with zero delay which can cause incorrect recovery times to be reported. As a remedy, in our work, we incorporate L2 Linux bridges [11] that are transparent to both the controller and the switches to design the failure experiment scenarios. We demonstrate that our approach more realistically emulates network failures even in Mininet.

In this thesis, we focus on the change of network and application based metrics regarding QoS and QoE, respectively, in case a failure in the data plane. Moreover, we investigate the effect of congestions on QoE. Thus, main contributions of our study can be summarized as:

- We propose a novel dynamic protection method considering Quality of Alternative Paths (QoAP). In this case, in addition to the recovery time, we also consider the alternative paths for flows affected by the failures. This is crucial to enhance both QoS and QoE.
- We investigate how video quality and thus QoE would fluctuate when a failure occurs in the path that video flows pass through. Since related studies in the literature consider only the variations of network parameters regarding QoS, this study carries out a unique touch to this problem considering the change of application parameters.
- Studies using an emulation environment such as Mininet for SDN do not consider the broken link scenario that may emulate the real-world case and therefore trigger the failure by using a special Mininet command. In this study, to emulate this realistically, we deploy a Linux bridge that operates in Layer 2 and is completely transparent to the controller and switches in the network for particular scenarios in order to make a valid comparison and evaluation.
- We detect the congestion in the link layer operating BFD rather than the application layer metrics that is generally used by video applications to adapt themselves for new conditions. Detecting congestion in the network level through SDN allows us to implement a global solution as opposed to each client adopting themselves in a reactive manner.
- After its successful detection, we employ the data plane fault tolerance mechanism to solve the congestion problem for DASH in SDN. As a result, we apply the same treatment carrying out for the link failure, to a different problem, congestion, that shows similar symptoms.

- We explore the impact of BFD intervals on the QoE parameters along with the video segment size to come up with a feasible operational range where the fault tolerance approach proves useful for the congestion case.

The rest of this thesis is organized as follows. Section 2 provides the background information about SDN, fault tolerance in SDN data plane, video streaming, and QoE. In Section 3, we elaborate the related works including DASH, fault tolerance, and congestion studies that use SDN as their networking paradigm. We explain the details of our fault tolerant design in Section 4. Afterwards, in Section 5, 6, and 7, we present the performance evaluation of our system along with the experiments. Finally, we conclude this thesis in Section 8.

2. BACKGROUND

2.1. Software-Defined Networking

Software-Defined Networking (SDN) is a paradigm in which control and data planes are separated. In SDN, the control plane is logically centralized and the data plane consists of basic forwarding devices comparing to traditional networking as shown in Figure 2.1. Its central view and programmability feature using the OpenFlow protocol [12] allow developers to implement third-party network applications that run on top of the controller, which is technically achieved via the Northbound Interface [13–15]. Thus, SDN makes possible to apply those applications throughout the network without considering every single device and it provides to solve problems that are hard to achieve in traditional networks such as fault tolerance seamlessly.

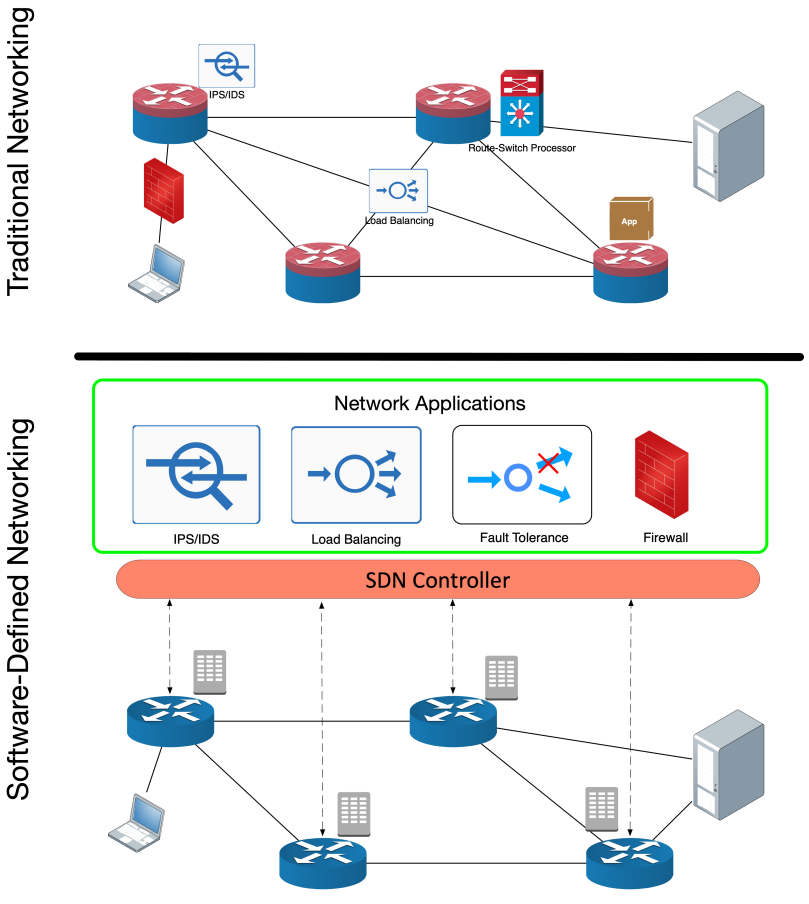


Figure 2.1. Traditional networking vs SDN

In traditional networks, applications including firewalls, congestion detection systems, and load balancers are configured separately since each device in the network may have different requirements in terms of software/hardware and may be manufactured by different vendors. Thus, designing a new application in a traditional network is complex and therefore needs a careful feasibility study. In SDN on the other hand, since it is programmable and logically centralized, new solutions and applications can be applied for the whole network easier than the traditional networks without performing such an effort. As a result, it offers greater control of a network through programming. To achieve this, the controller pushes the appropriate rules to the flow-tables of suitable forwarding devices considering the requirements of each application. Thus, all applications can take advantage of the same network information that is called as the global network view.

The communication between the controller and forwarding devices in SDN is performed using the southbound API. Currently, the OpenFlow protocol, which provides an open and standard way for a controller to communicate with a switch, is used as a de facto standard for this purpose. OpenFlow provides an open protocol to program the flow-tables in different switches and routers. Thus, researchers can control their own flows. On the other hand, there is no such standard for communication between the controller and applications. This interface is called the northbound API.

2.2. Fault Tolerance in SDN Data Plane

As far as fault tolerance is concerned, the SDN data plane shows similarities with traditional networking as faults in this plane consist of the switch and link failures. However, SDN has important advantages for dealing with fault tolerance using the centralized view possessed by the control plane elements. In order to achieve a fault tolerant data plane, two steps need to be carried out respectively: (1) failure detection and (2) path calculation. Considering these two steps, the failure recovery time, T_r, can be computed using the failure detection time, T_d and the path calculation time, T_{pc}, as given in Equation 2.1.

$$T_r = T_d + T_{pc} \tag{2.1}$$

2.2.1. Failure Detection

Detection of the failure is the most critical step of fault tolerance since all recovery methods are bounded by the failure detection time. Depending on the network traffic load, this duration varies between milliseconds and seconds. There are three methods in SDN to be used for failure detection:

2.2.1.1. Loss of Signal (LoS). When a port of the SDN-based switch fails, link failure can be detected by using LoS and the problem is notified to the controller as an event.

2.2.1.2. Link Layer Discovery Protocol (LLDP). LLDP packets are used in SDN to discover the topology as done in traditional networks. To perform this, the controller periodically commands switches to flood LLDP packets through all of their ports. If a link or multiple links in the topology fail, the controller can detect the failure by noticing the missing LLDP packets in seconds since there would be no response from the formerly working device.

2.2.1.3. Bidirectional Forwarding Detection (BFD). BFD protocol can be run in computer networks with any transport protocol for fault tolerance since it is protocol-independent. BFD is used for paths between two nodes in order to observe failures and disruptions in communication quickly. These two nodes can be connected either directly or through multiple hops. Each node transmits a control packet including the current state of the monitored link or path to its pair node. When a node receives the control message, it sends an echo message with the session status. Accordingly, failure detection time T_d is computed based on the message transmit interval T_i and the detection time multiplier M as given in Equation 2.2. The detection time multiplier is

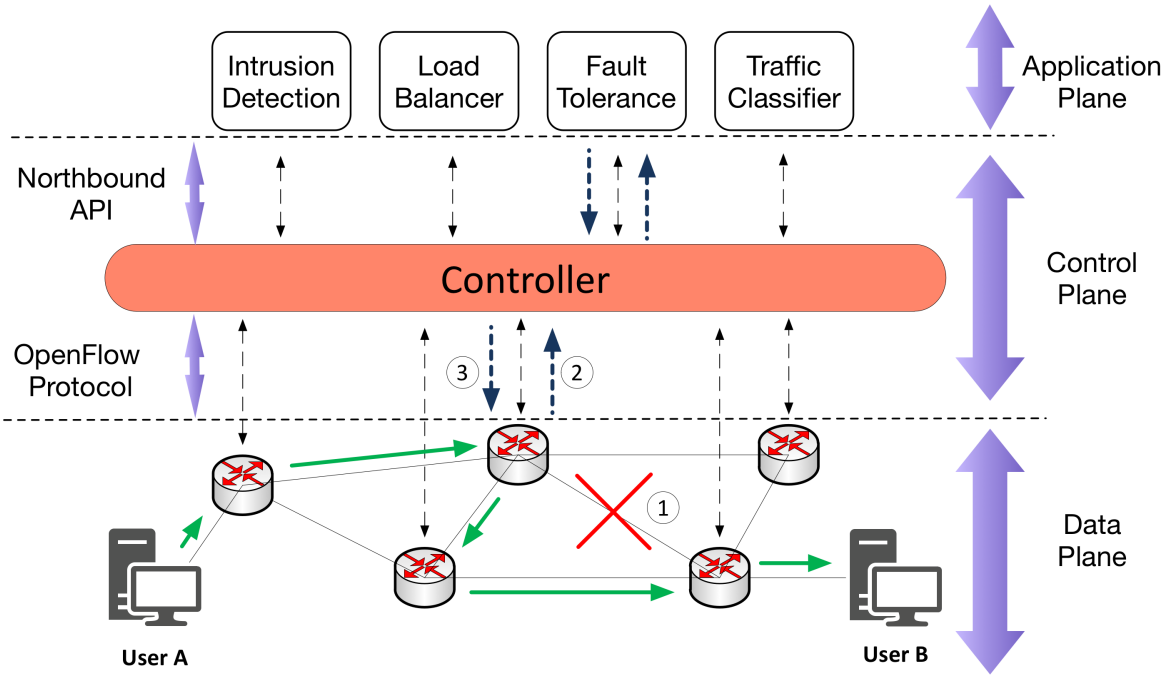


Figure 2.2. Restoration in SDN.

used to prevent false positives that can occur due to packet loss.

$$T_d = (M + 1) * T_i \tag{2.2}$$

2.2.2. Path Calculation

After the failure detection, path calculation should be handled to recover from the failure state. Restoration and protection are two essential approaches for the failure recovery in the data plane.

2.2.2.1. Restoration. Restoration is a reactive approach for path calculation because the secondary path is computed after the failure of the primary path. During this process, the controller itself is the responsible entity as shown in Figure 2.2. When a failure occurs in the data plane, it is initially detected by the corresponding switch and

then an event is generated for the controller to trigger the path calculation process. Afterwards, the event is processed in the northbound application and the new route is computed. Finally, appropriate flow rules are sent to the relevant switches in the data plane to maintain the communication. Thus, Equation 2.3 including the propagation time, T_p, between the controller and corresponding switches, switch update time, T_{update}, for changing flow-rules can be derived from Equation 2.1.

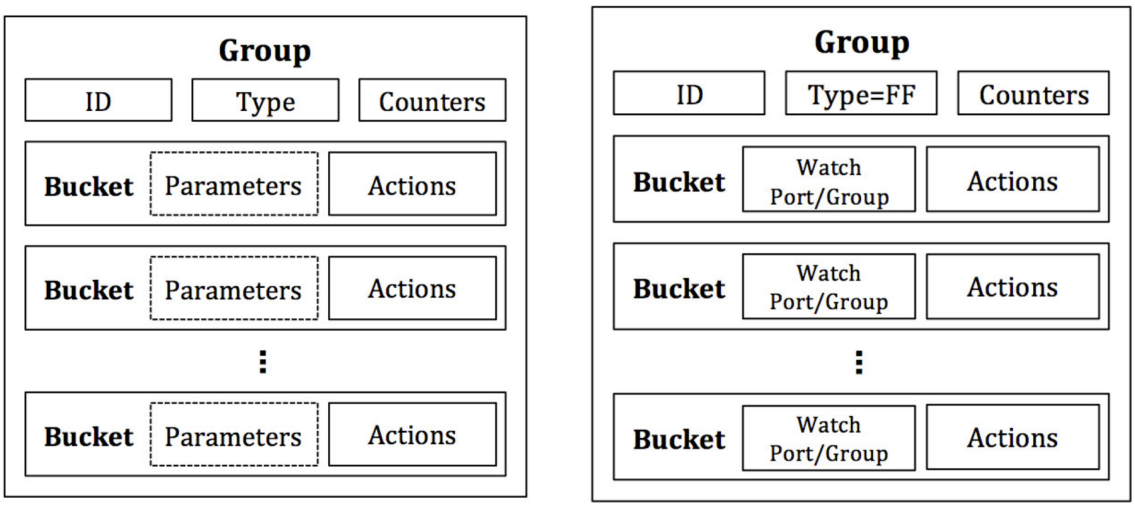
$$T_r = T_d + T_p + T_{pc} + T_p + T_{update} \tag{2.3}$$

2.2.2.2. Protection. The protection approach is a proactive way to handle the data plane failures since the secondary path(s) are calculated before any occurrence of the fault. Moreover, thanks to the Fast Failover groups proposed in OpenFlow version 1.1, the controller involvement is not necessary to update the flow rules in case of a failure. Thus, the failure recovery time is shorter than the restoration in this approach as given in Equation 2.4.

$$T_r = T_d + T_{update} \tag{2.4}$$

2.2.3. OpenFlow Groups

In the first stable version of OpenFlow, namely 1.0, there was no specific functionality for fault tolerance. Therefore, network managers and researchers used their own techniques to overcome the failures in SDN until OpenFlow version 1.1 in which the group table concept was introduced. The goal of the OpenFlow groups is to apply specific operations that cannot be defined by the flow itself such as backup path information on packets. A group consists of entries that have a group identifier, a group



(a) A Standard OpenFlow Group Entry. (b) OpenFlow fast failover group.

Figure 2.3. OpenFlow Groups.

type, counters, and a list of action buckets as shown in Figure 2.3(a). One of the most important features brought by OpenFlow groups is the ability to define multiple lists of actions, which is called an action bucket, for each group entry. This feature makes possible to perform various traffic engineering operations in SDN. When a packet matches with a flow rule, it is assigned to the appropriate group entry. There are currently four OpenFlow group types:

- *All Group:* This group is used for multicasting or broadcasting. A packet in this group is copied for each bucket in the bucket list and handled independently.
- *Select Group:* It is developed for load balancing. The packet in the group entry is sent to a single bucket. Determining the corresponding bucket is performed by the switch itself with a selection algorithm such as round robin.
- *Indirect Group:* There is only one bucket in this group type. The goal is to cover commonly used actions for the same next hop when forwarding and thus reduce the switch memory utilization.
- *Fast Failover Group:* Likewise, in the select group, the packet is sent to only a single bucket. The difference is that there is no bucket selection in this group. The packet is sent to the first live bucket. The liveness of bucket is checked by *watch port/group* parameters. The schema of this group is depicted in Figure 2.3(b).

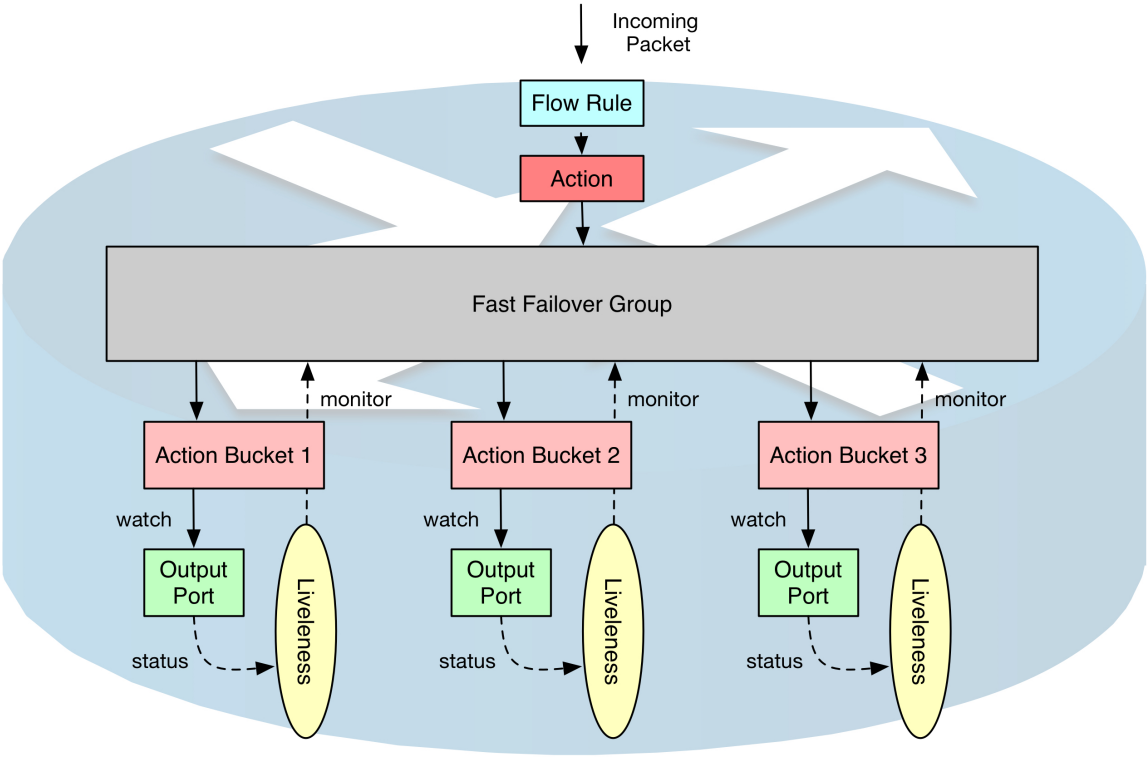


Figure 2.4. The working mechanism of a Fast Failover group in OpenFlow switch.

Fast Failover groups are used for the protection approach in SDN. They provide alleviated control path traffic since the controller is not involved in the failure, and reduced recovery time as the problem is solved in the data plane. The working process of Fast Failover group concept is depicted in Figure 2.4. The incoming packet is first evaluated by the flow-rule table and then the corresponding action, which is Fast Failover group in this case, is applied. Afterwards, based on the liveliness status of the possible output ports that are monitored continuously, the packet is sent to the first available egress port. Thus, if a link fails, the appropriate action is instantly applied by the switch without consulting the controller.

2.3. Video Streaming and Quality of Experience

Video streaming has dramatically changed for the last few years due to the recent developments of the Internet technologies and protocols. Considering the increasing usage of mobile devices and the high-resolution options, this change is inevitable. In traditional video streaming, protocols like Real-Time Streaming Protocol (RTSP) be-

have in a stateful manner via tracking the state of the clients during the streaming. Moreover, when a streaming process has been started between a client and a server, the connection is maintained as the stream of packets until the video file is completely transferred. However, this approach is not sufficient today considering the dynamic needs of video streaming.

On the other hand, the Hyper-Text Transfer Protocol (HTTP) is stateless; when the client receives its requested data, the connection is terminated. Thus, the streaming is performed in a more dynamic manner when HTTP is used since each HTTP request results in a new transaction that provides many advantages for video streaming [4]. As a result, Dynamic Adaptive Streaming over HTTP (DASH) is proposed by addressing the weaknesses of the traditional streaming methods.

DASH operates upon fixed durations of video segments called "representations" which may belong to different bitrates. DASH introduces further adaptivity to the dynamic nature of HTTP streaming by enabling a switching mechanism among different representations. To perform this concept, first, the video file is sliced into the fixed timed parts, namely segments or chunks, at the server as described in a Media Presentation Description (MPD) file. These video parts generally vary from 1 second to 15 seconds of duration. The client can select appropriate segments among different representations based on its application metrics including the buffer level and throughput. Thus, a playback can typically consist of different representation segments instead of homogeneously defined video streaming files. This concept of DASH is depicted in Figure 2.5.

The most important advantage of DASH considering QoE is the switching mechanism among different representations since it prevents stalling and thus the client can continue to play the video under several conditions. To measure QoE, there are subjective and objective measurements. In subjective measurements, users vote their perception of video using the 5-point scale where 1 represents "poor" and 5 represents "excellent". This process is named as Mean Opinion Score (MOS). On the other hand, in objective measurements, several QoE metrics for DASH [16] including HTTP

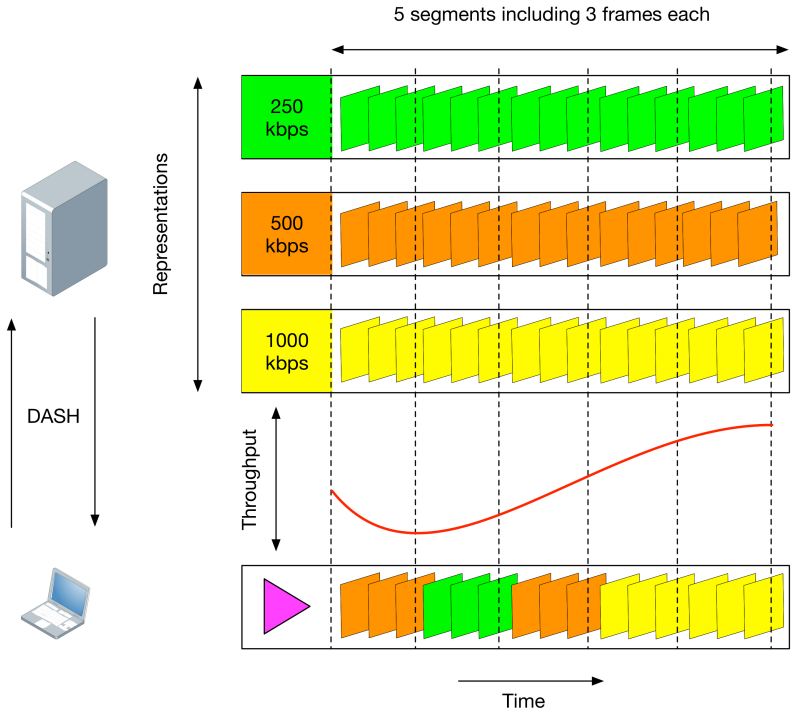


Figure 2.5. Concept of DASH

transactions, representation switch counts, buffer level, and bitrate are evaluated and finally, a formula is generated. Actually, since the video quality is the most distinctive component of QoE, many studies used the video quality as the objective measurement for QoE.

One of the most used metrics to measure the quality of the video is the Structural Similarity Index (SSIM) [17]. It is originally used to measure the quality of an image by comparing it with the original image. Since a video consists of multiple images, the same technique is widely used for measuring the video quality. Thus, mapping the video quality and bitrate is coherent because higher bitrate provides higher similarity with the original video. However, as shown in Figure 2.6, the mapping of the bitrate to the video quality using SSIM demonstrated that the relationship between bitrate and perceptual quality is not linear [18]. Hence, using three resolution types and various bitrates given in Table 2.1, a generalized function for QoE based on bitrate and resolution is configured by conducting curve fitting [18]. The formula is given in Equation 2.5 and corresponding coefficients are given in Table 2.2. In the equation, *f(x)* represents the video quality, and variable x denotes the bitrate.

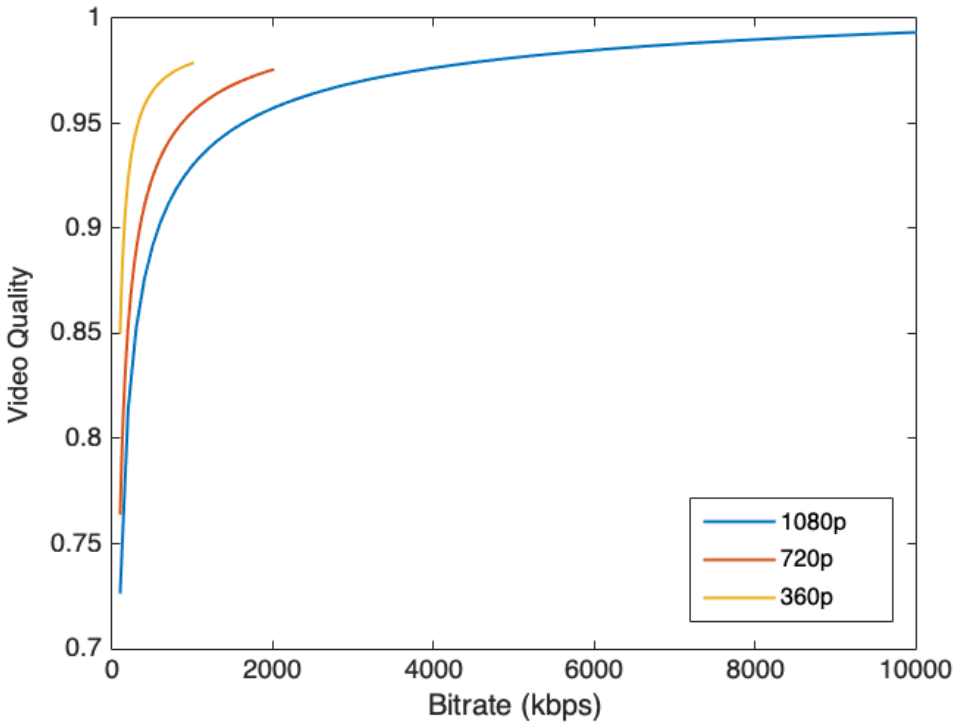


Figure 2.6. Relation between video quality and bitrate for three different resolutions.

Table 2.1. Bitrates for different screen resolutions

Resolution	**Bitrate (kbps)**
1080p	100, 200, 600, 1000, 2000, 4000, 6000, 8000
720p	100, 200, 400, 600, 800, 1000, 1500, 2000
360p	100, 200, 400, 600, 800, 1000

Table 2.2. Coefficients of the generalized video quality function

Resolution	**Power Series Model**			**Goodness of Fit**	
	a	b	c	Adjusted R^2	RMSE
1080p	-3.035	-0.5061	1.022	0.9959	0.006011
720p	-4.85	-0.647	1.011	0.9983	0.002923
360p	-17.53	-1.048	0.9912	0.9982	0.002097

$$f(x) = ax^b + c \tag{2.5}$$

3. LITERATURE REVIEW

SDN, with its dramatically different paradigm, offer many benefits to address networking challenges that are hard to circumvent with traditional approaches. The existence of a centralized controller in addition to the radically different flow of events provided by SDN render these solutions technically feasible. One such challenge is the fault tolerance including data plane, control plane, and application plane domains. Fault tolerance in each domain leads to different problem formulations. This thesis focus on the data plane domain as it is the level where the actual traffic is handled.

3.1. Restoration Approaches

Considering fault tolerance in SDN, one of the most important concepts in OpenFlow protocol is the Fast Failover (FF) groups that allow for solving faults in the data plane without involving the controller. However, before OpenFlow protocol version 1.1 in which OpenFlow Groups including Fast Failover groups are introduced, many studies [19–22] that work on this topic used the restoration approach. The common behavior in these studies is that they used the controller for failure notification and then calculate new routes to maintain communication. Kim *et al.* [20] used VLANs to compute routing paths. In [19], Sharma *et al.* compared their fast failover system with the Learning Switch, Learning PySwitch, and Routing Mode of the NOX controller [23]. Nguyen *et al.* [21] used SDN for fault tolerance in Wide Area Networks (WANs) since routing protocols such as BGP and OSPF undergo serious problems in failures. For example, while BGP has prolonged route convergence time, OSPF has long recovery time. Li *et al.* [22] on the other hand developed a restoration approach by using a local optimal failover method in order to reduce the path calculation time. Some studies also considered reliability in this manner. In [24] Yuan *et al.* designed a system based on the Byzantine model to tolerate faulty switches in order to enhance reliability. Moreover, Song *et al.* [25] focused on control-path reliability which is an important consideration for out-of-band controllers whose network view can be affected by the data plane failures.

3.2. Protection Approaches

Since using the controller for fault tolerance increases the recovery time, many studies proposed methods to solve this problem in the data plane without involving the controller. In [26], authors built a system that permits switches to send faulty link information to only the relevant switches in order to prevent traffic flood and increase the network performance in centralized networks. Thus, when a link fails in the network, the corresponding switches create Link Failure Messages (LFM) and send them to the relevant switches. Their experiments showed that switches are informed of the failed link sooner than the controller identifies and commits an update. Likewise, Kempf *et al.* [27] supported a monitoring function for failure detection in the data plane without involving the controller. To this end, they generated monitoring messages within the source switch and process them in another destination switch. If the destination cannot receive these packets for a long enough period, their method concludes that there is a fault in the current path. Their experiments showed that the data plane fault recovery can be achieved in a scalable way within 50 milliseconds using this function. In [28], Ramos *et al.* extended their previous study [29] and developed a proactive failure recovery scheme by carrying the information of alternative paths in the packet headers. Thus, when a link failure happens, their system uses alternative path information in order to maintain communication without consulting the controller. In the packet header, they used VLAN and MAC Ethernet fields to carry alternative paths.

On the other hand, Reitblatt *et al.* [30] proposed a new language based on Regular Expressions for implementing fault-tolerant network programs in SDN by using OpenFlow Fast Failover groups. They allowed developers to specify the set of paths that packets may take through the network as well as the degree of fault tolerance required. Thus, their compiler generated rule-tables and group-tables that provide specified fault tolerance. Accordingly, Petroulakis *et al.* [31] proposed a pattern framework for fault tolerance using a rule-based language. Moreover, Cascone *et al.* [32] used finite state machines in the data plane for fast failure detection and then recovery.

After OpenFlow protocol version 1.1 was introduced, studies used Fast Failover groups as it provides many benefits in fault tolerance including recovery time and control-path traffic. To this end, Sharma *et al.* [8] considered carrier-grade networks in which fault recovery should be completed in 50ms, and therefore they performed protection mechanisms using OpenFlow Fast Failover groups. The experimental results showed that the protection approach diminishes the time required for fault recovery and mitigates the traffic load on the controller. Moreover, in [33], they focused on failure recovery for the in-band OpenFlow networks in which control and data traffic are transmitted on the same channel by applying the same scenarios. In [34], Borokhovich *et al.* implemented traditional graph algorithms including BFS, DFS, and Module to compute backup paths which are used in FF groups. Adrichem *et al.* [35] used Bidirectional Forwarding Detection (BFD) protocol [36] per link as well as [8] to detect failures and then compared the performance of different BFD detection intervals in terms of milliseconds. Pfeiffenberger *et al.* [37] on the other hand, focused on the robust multicasting in SDN by considering fault tolerance. In [38], the authors used VLAN tags in their design in order to reduce alternative path rules.

3.3. DASH and QoE in SDN

Recently, studies working on DASH use SDN as the main networking technology in order to exploit the benefits of the centralized controller and, thus, enhance QoE. Zabrovskiy *et al.* [39] applied DASH in Mininet and compare its performance with a specialized hardware-software emulator using the same channel characteristics. The performance results obtained from both settings were comparable, which enabled authors to conclude that Mininet can be used as a reliable emulator for DASH. Mkwawa *et al.* [40] proposed a video quality management scheme that considers the traffic intensity. They compared the number of stalls of the DASH video streaming when their proposed scheme is used and not used. As expected, the number of stalls were fewer when they used their solution. In [41], Bentaleb *et al.* focused on QoE unfairness for multiple clients in the network since when the number of clients increases, QoE is unfair because of bandwidth sharing and network resource underutilization. Afterwards, they

improved their scheme in [42], considering scalability issues of clients, communication overhead, and client heterogeneity. Likewise, Bagci *et al.* [43] studied QoE fairness among clients. They used the network slicing concept and manipulated TCP windows to prevent QoE fluctuations.

3.4. Congestion in SDN

SDN is also used for the congestion case by several studies. In [44], the authors modified the TCP receive window of ACK packets at the controller in order to avoid network congestion. To perform this, they deployed a queue management scheme in OpenFlow-switch that notifies the controller when the queue passes the given threshold. Kim *et al.* [45] on the other hand considered the dynamic changes in the network traffic and proposed their reinforcement learning based technique, Q-learning, for the routing of flows to prevent congestion. Cheng *et al.* [46] indicated the shortage of ternary content addressable memory (TCAM) of OpenFlow switches that cause a bottleneck for scalable flow management. Therefore, they applied flow aggregation using VLANs to prevent congestion for a failure recovery case. In [47], the authors carried out a congestion control mechanism in Mobile Edge Computing (MEC) environment using SDN considering congestion. They classified the network traffic as delay tolerant and delay sensitive so that they buffered the delay tolerant flows in MEC servers during the peak hours in order to prevent congestion.

To the best of our knowledge, there is no study in the literature that examines QoE for fault tolerance in SDN considering video streaming using the DASH paradigm. Moreover, studies worked on fault tolerance considered only QoS parameters including recovery time, delay, and the number of affected flows rather than application parameters that affect QoE. On the other hand, this is the first study that applies a fault tolerance approach to improve QoE in case a network-based problem like congestion. Thus, our study is distinctly separated from the existing works.

4. DESIGN OF THE FAULT TOLERANT DATA PLANE

We implemented three modules as northbound applications using the Floodlight controller [48]. The first module is restoration, which is the reactive way to handle the data plane failures. The second and third modules are protection approaches which are named as static and dynamic protection, respectively. In the static protection module, backup paths are calculated before the failure occurs and they remain fixed even the network conditions change. On the other hand, we combine the active nature of restoration and responsive feature of static protection in the dynamic protection method.

Apart from the northbound applications, we also developed a DASH client module to evaluate the metrics of video streaming in the case of a failure. In order to perform an accurate evaluation of the DASH application, the operations and functionalities of the DASH paradigm are presented along with a discussion on QoE measurement.

4.1. Restoration Module

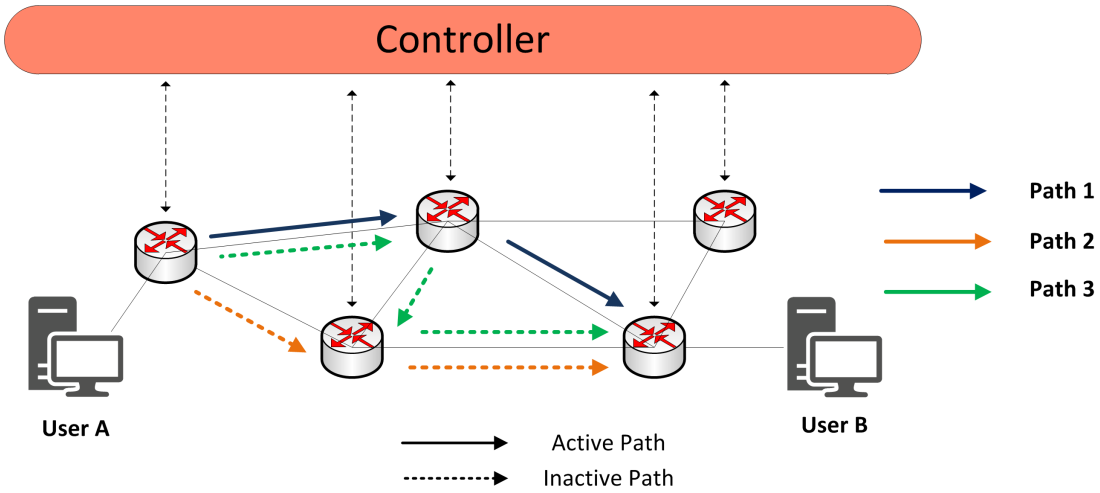
In the restoration, as explained in Section 2.2.2.1, the essential idea is the involvement of the controller. Because of the dynamic nature of the network environment, the restoration approach is important to handle the failures in the network. We implemented the restoration module as demonstrated in algorithm given in Figure 4.1 by applying the steps depicted in Figure 2.2. First, the failed link information is received by the controller as an event/input. Based on this information, the current network state is computed in order to calculate the new route based on the smallest hop count for the flows affected by the failure. Afterwards, the affected paths are identified via the active flow pool using the failed <switch, port> tuple and then a new path is calculated for each flow. Finally, old rules are replaced with the new ones determined by the restoration module.

```
Input: Failed link information including its src/dst switch and port numbers
Output: The new route
linkInfo = ExtractLinkInfo(failedLink);
ComputeCurrentNetworkState();
for path ∈ activePaths do
    if failedLink ∈ path then
        newPath = ComputeNewPath(failedLink.src, path.dstSwitch);
        removePathFromSwitches(path);
        InstallRules(newPath);
    end
end
```

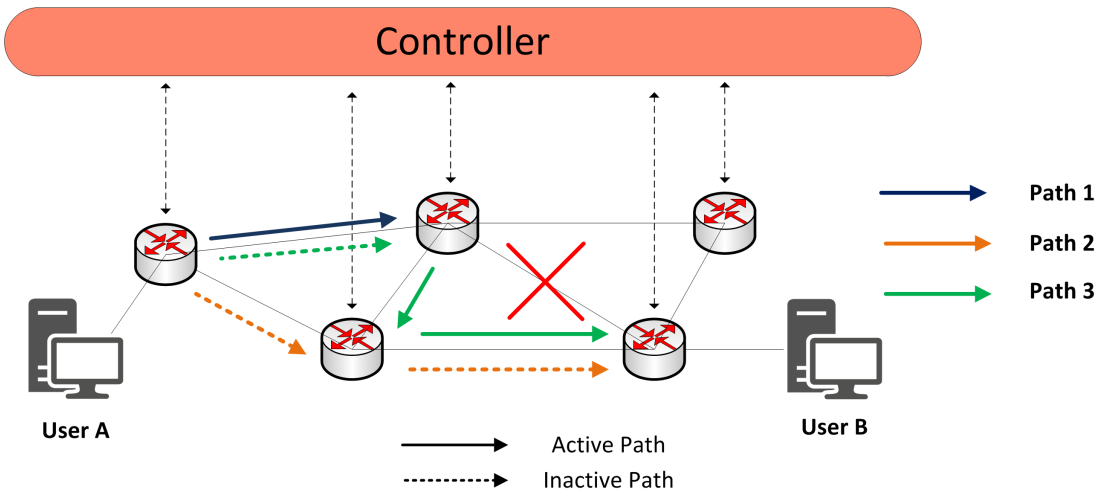
Figure 4.1. The Algorithm of Restoration Module

4.2. Static Protection Module

We used OpenFlow Fast Failover groups for this module to apply backup paths in case a failure occurs in the primary path. Thus, the failure is recovered without the involvement of the controller and the burden on the control plane and control channels is mitigated as depicted in Figure 4.2. First, all possible routes between the new source-destination pair are calculated and one of them is selected as the primary path based on the hop count. Afterwards, if a failure occurs at one of the resources on the primary path, Fast Failover groups handle it using the working buckets in which the backup actions are defined. As shown in Figure 4.2(b), the primary path is actually used to forward packets until the switch to which the failed link is connected. However, since the watch/port group belonged to the primary path in that switch cannot work for the rest of the route because of the failure, the secondary path would become active by forwarding the packets of flows to the corresponding switch. Moreover, since the whole operation after the failure is carried out in the data plane, the burden on the controller would significantly reduce.



(a) Primary and backup paths between communicators.



(b) Static protection using OpenFlow groups.

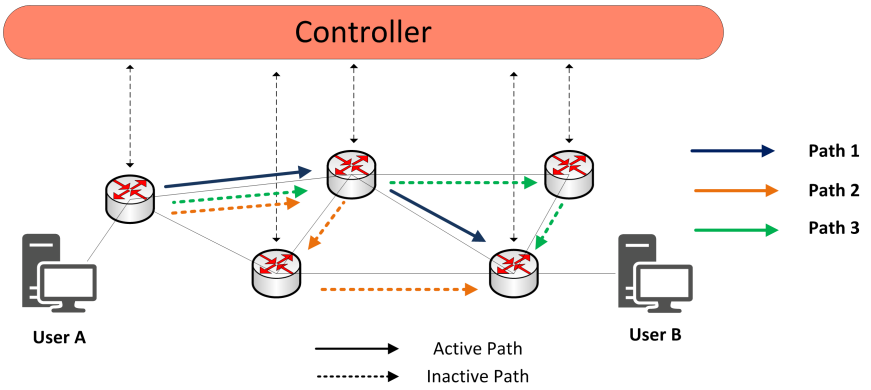
Figure 4.2. Static Protection in SDN.

Since the controller is not involved for the failure in the protection approach, this module considers only the installation of the flow rules including the primary and backup paths on the data plane in a proactive way. Hence, the algorithm given in Figure 4.5 shows only the steps that how the routes are calculated initially and then the configuration with the installation of the Fast Failover groups. First, the current network state is computed for effective routing. Second, for a given source and destination pair, all possible paths between them are calculated and then assigned to the $allPaths$ variable. Afterwards, for each path in this set, flow rules are installed into the appropriate switches in the data plane by using $InstallProactiveRules$ function. In this function, since a path consists of <switch, port> tuples in our design, we investigate each switch in the path considering that whether it has the bucket that directs flows to the corresponding egress port or not. If it has buckets for given $groupID$, which indicates the group of paths between a <source, destination> tuple, and there is no egress port information in the bucket, we append it and then update the switch. On the other hand, if there are no buckets, we create one with the egress port information for the switch. Thus, all paths are installed into the data plane as the proactive procedure.

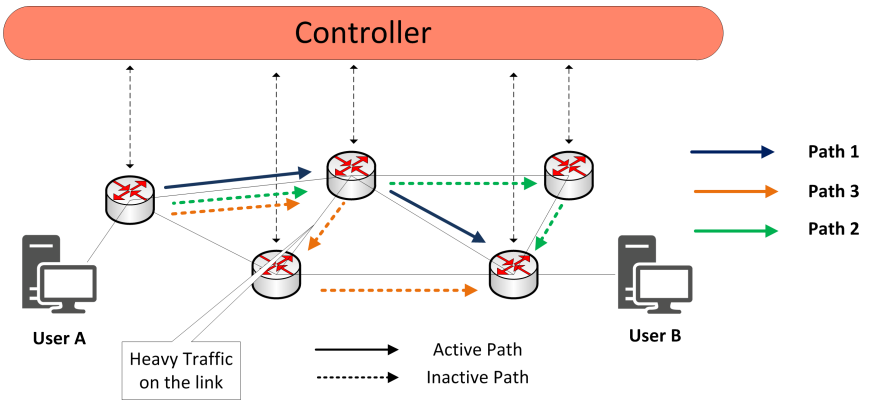
4.3. Dynamic Protection Module

The static protection has many benefits over the restoration as the failure is handled in the data plane. However, the performed action using the Fast Failover groups for the failure is actually based on the network information that is taken at the beginning of the communication. Therefore, since the network environment can change over time because of various reasons, the applied actions may not be efficient for the communication even though it provides the continuity of it. Thus, in this module, we combine the advantages of the restoration module in which the actions are applied using the recent condition of the network, and the static protection module which provides responsiveness for the failures without involving the controller.

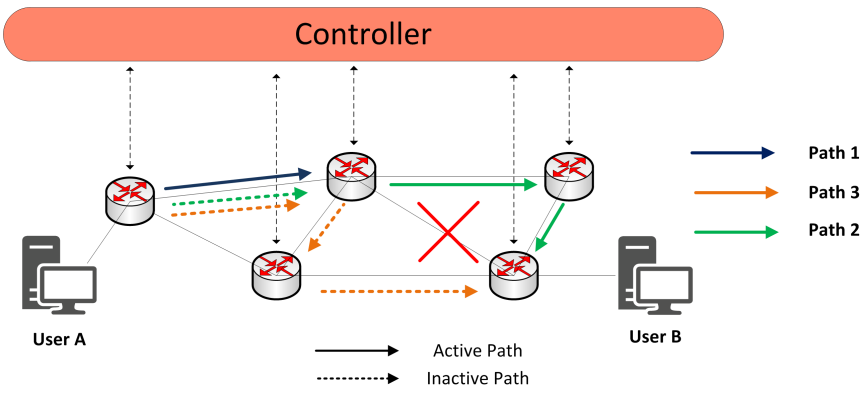
The main task of this module is to find the best alternative path based on the latency parameter. Since Floodlight provides an API call to obtain the latency values



(a) Primary and backup paths between User A and User B.



(b) Heavy traffic on the original Path 2 and then changing the order of the backup paths.



(c) Evading the failure in the network considering the quality of the alternative paths.

Figure 4.3. The concept of the dynamic protection module.

of the links, this information is used for each path by determining the best alternative path for every QoAP calculation interval, T_{qoap}. A demonstrative example is shown in Figure 4.3. After the formation of the primary path and two backup paths as depicted in Figure 4.3(a), the secondary path, Path 2, becomes loaded with a recently generated heavy traffic as shown in Figure 4.3(b). This situation is detected by our dynamic protection module and subsequently, Path 3 is replaced as the secondary path since its latency value is currently smaller than Path 2. Finally, as shown in Figure 4.3(c), when the link of the primary path fails, the quality of the alternative path has already been considered and therefore communication continues with acceptable quality.

We implemented this module using the algorithm given in Figure 4.6. First, the current network state is analyzed in order to get the recent information of the links, switches, and their load conditions. Afterwards, for each $groupID$, which indicates a <source, destination> tuple, all computed paths are examined and then the primary path is extracted by checking whether the path is currently active or not. Afterwards, each primary path and its corresponding $groupID$ is given as the parameters to the function $OrganizeBucketList$ in which the best alternative path is calculated from each <switch, port> tuple that is in the primary path and then bucket lists are reorganized. This operation is repeated based on the QoAP calculation interval, T_{qoap}, that must be configured for the network conditions.

4.4. BFD-based Congestion Detection Module

We designed this module based on the data plane fault tolerance mechanism applying the restoration approach rather than protection. Since video streaming can resist network changes for seconds through its buffering mechanism, rerouting flows based on the recent network view would be beneficial. Similar to the fault tolerance problem, our BFD-based Congestion Detection Module includes a detection phase and the action phase. However, since there is no need to reroute all affected flows in the congestion problem, it is separated from the fault tolerance. Figure 4.4 shows the main aspects of our design.

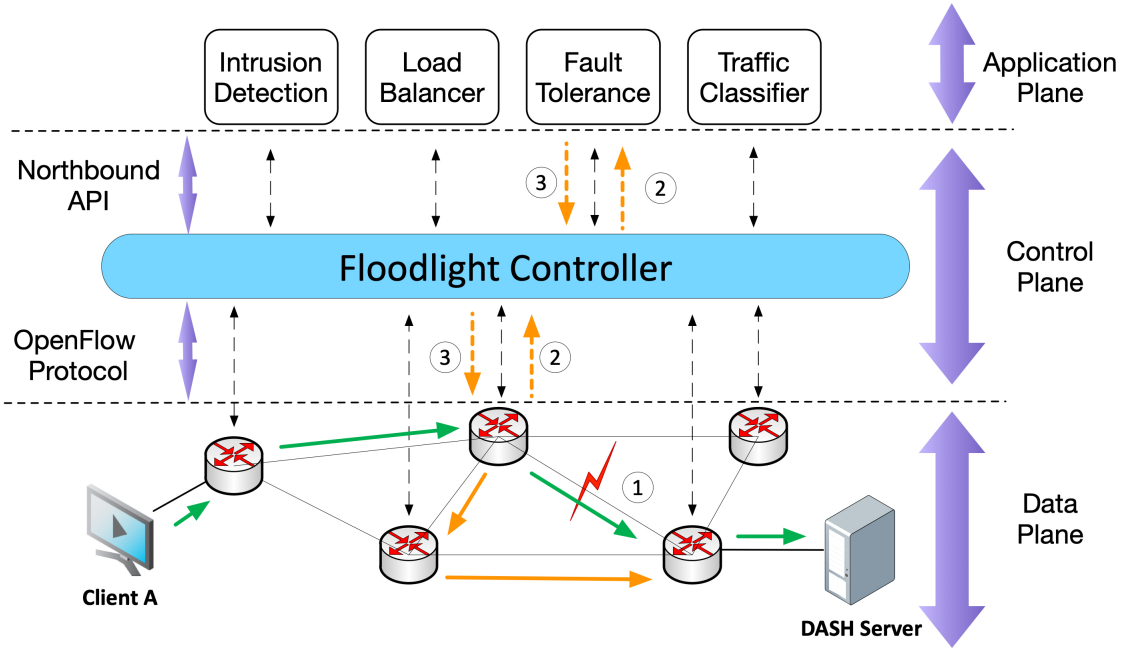


Figure 4.4. The design of the BFD-based Congestion Detection System.

4.4.1. Congestion Detection

The first step is the detection of the congestion using the BFD in the case of a heavy traffic load as shown in Figure 4.4. BFD is run on the switches to which the observed link is connected. These two switches are called as a pair regarding to BFD. Each switch conveys a control packet including the current state of the monitored link to its pair switch. When a switch receives the control message from its pair, it sends an echo message to it with the session status.

The failure detection time, T_d, is based on the BFD message transmit interval, T_i, that can be manually configured by the network administrator considering the given services. For example, if real-time applications such as VoIP are widely used in the network, the interval must be very low considering the 50ms recovery time [8]. On the other hand, if there is multimedia traffic like video streaming, the interval may be in seconds due to the buffer mechanism of the applications. Thus, T_d is computed based on the T_i and the detection time multiplier M as given in Equation 2.2. M value is used to prevent false positives.

4.4.2. Rerouting Flows

After detecting the congestion using BFD, the problem on the link is reported to the controller via the OpenFlow protocol. Subsequently, the controller informs the modified fault tolerance application about the problem. In the application, flows passing through that link are first extracted from the flow pool in which all active flows are held. Afterwards, the predefined percentage of flows are selected for rerouting and the new rules are created for them. As a result, the congested link is relieved.

4.5. DASH Module

To calculate the necessary metrics including the buffer level and the bitrate for the video quality value using Equation 2.5, we implemented a video client module using DASH.js [49]. It is a widely used Javascript library to measure the DASH client metrics. Thus, this module consists of several functions for observing the changes in the metrics in the case of a failure.

Apart from the failure, for the congestion scenario that includes multiple clients, each DASH client reports their QoE parameters including the bitrate, video quality value, latency, and the number of quality switches for the evaluation of the effect of congestion. All these parameters except the video quality value are extracted from the DASH.js API.

```
Input: <source, destination> tuple
Output: Primary and backup paths in the data plane using Fast Failover
        groups
Function BuildingProtection()
    ComputeCurrentNetworkState();
    allPaths = ComputeAllPaths(src, dst);
    groupID++;
    for path ∈ allPaths do
        InstallProactiveRules(path, groupID)
    end
end
Function InstallProactiveRules(path, groupID)
    index = path.length();
    for switch ∈ path do
        outPort = path.get(index);
        key = getKey(switch + groupID);
        buckets = getBucketList(key);
        isBucketsExist = false;
        if buckets != null then
            isBucketsExist = true;
        end
        if isBucketsExist = false then
            buckets = createBucketList();
        end
        if buckets.contain(outPort) = false then
            buckets.add(outPort);
        end
        switch.update(buckets);
    end
end
```

Figure 4.5. The Algorithm of Static Protection Module

```
Input: Paths of groups, bucket list for groupIDs
Output: The Best backup path based on latency
Function EvaluationOfPaths ()
    ComputeCurrentNetworkState();
    for groupID ∈ groupIDs do
        groupPaths = pathsOfGroups(groupID);
        if groupPaths ≠ null then
            for path ∈ groupPaths do
                if IsPathActive(path) = true then
                    primaryPath = path;
                    break;
                end
            end
            OrganizeBucketList(groupID, primaryPath);
        end
    end
end
```

Figure 4.6. The Algorithm of Dynamic Protection Module

5. PERFORMANCE EVALUATION OF NETWORK-BASED METRICS

In this chapter, we evaluate the network-based metrics including the recovery time and packet loss that are important metrics for QoS in fault tolerance. We conducted several experiments using iPerf [50] and Wireshark [51] tools on several scenarios including the restoration, static protection, and dynamic protection modules. To this end, we first carried out a comparison between the recovery time and packet loss in order to observe their relationship for the case that the packets are produced for each millisecond. Afterwards, we assessed the effect of the number of flows and different topologies on the recovery time in terms of packet loss. Then, we focused on the creation of the link failure deploying Linux bridge in Mininet to observe its impact on our fault-tolerant modules. Accordingly, we analyzed the effect of BFD intervals on the recovery time. Finally, we evaluated the importance of the alternative paths for the data plane fault tolerance.

We repeated experiments 10 times the variance control. We used Mininet for the SDN emulation, and Open vSwitch [52] for virtual switches created in Mininet. In the experiments, based on the scenario, we trigger the failure in two ways:

(i) *Using Mininet Command:* We used the standard Mininet command, as *link switchA switchB down*, to create the link failure in a given topology. However, in addition to the link, this command also destroys the ports of switches to which the link is connected. Since studies that investigate the fault tolerance problem in the data plane using Mininet execute this command for the failure, we also applied it in our experiments.

(ii) *Using Linux Bridge:* In the real world, failures on the link usually happen without affecting the ports of the switches like a broken cable. This is crucial since affected ports cause an immediate notification for the switch related to the failure. Thus, to emulate a real world like failure event in Mininet, we shut down one of the

ports of the Linux bridge that is placed between the corresponding switches.

5.1. Recovery Time vs Packet Loss

The examination of the relationship between the recovery time and packet loss is important since observing the packet loss as a QoS metric is more convenient than the recovery time that requires additional tools or measurements. Therefore, our main consideration for this comparison is that whether we do achieve the same recovery time value measured by the Wireshark tool using the packet loss metric or do not. To this end, we used the restoration and static protection modules for the comparison by applying both Mininet command and Linux bridge command to create the failure and observe their performance. Considering the Linux bridge command, we used the topology depicted in Figure 5.1 while we employed the topology shown in Figure 5.2 for the usage of the Mininet command.

The results shown in Figure 5.3 indicate that if the transmission of the individual packets of a single flow can be configured as one packet per second, the packet loss would be used as a substitute for the recovery time. On the other hand, the results verify that the static protection approach is superior to the restoration in terms of the recovery time and packet loss. Furthermore, we can observe that if we use BFD as the fault tolerance detection method in the restoration and static protection approaches, it provides less recovery time based on its message interval considering their counterparts. In Figure 5.3, the T_i is configured as 5ms so that the detection time is 15ms. Apart from its better results, since BFD can allow network administrators important flexibility via the configurable detection time, a fault tolerant system can be adapted for each network requirement.

5.2. Effect of the Number of Flows

The impact of the different number of flows on the approaches for a fault-tolerant system must be examined. To this end, we evaluated 10, 20, 40, and 80 number of flows using the topology shown in Figure 5.2. We created the failure on the link between

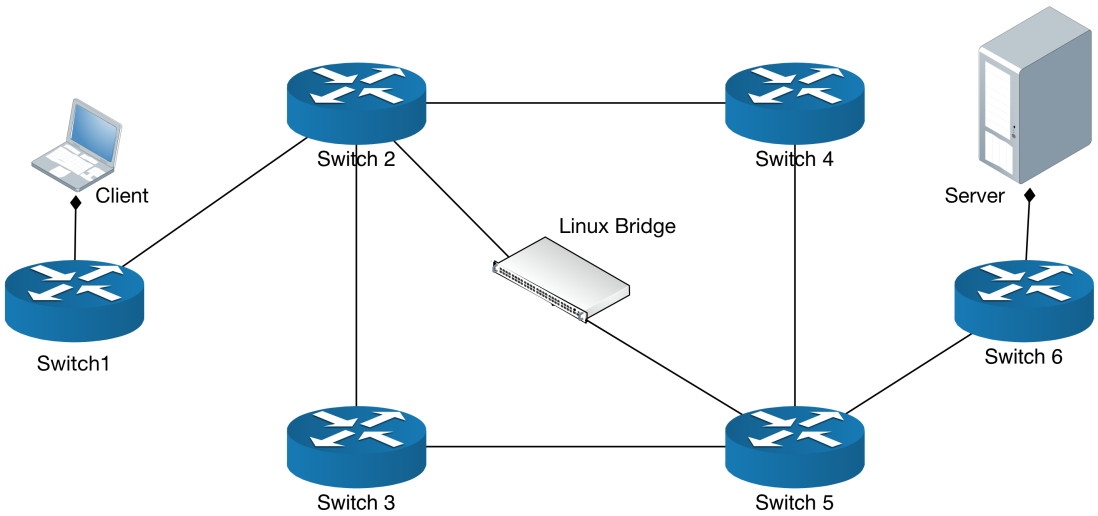


Figure 5.1. The topology using a Linux bridge.

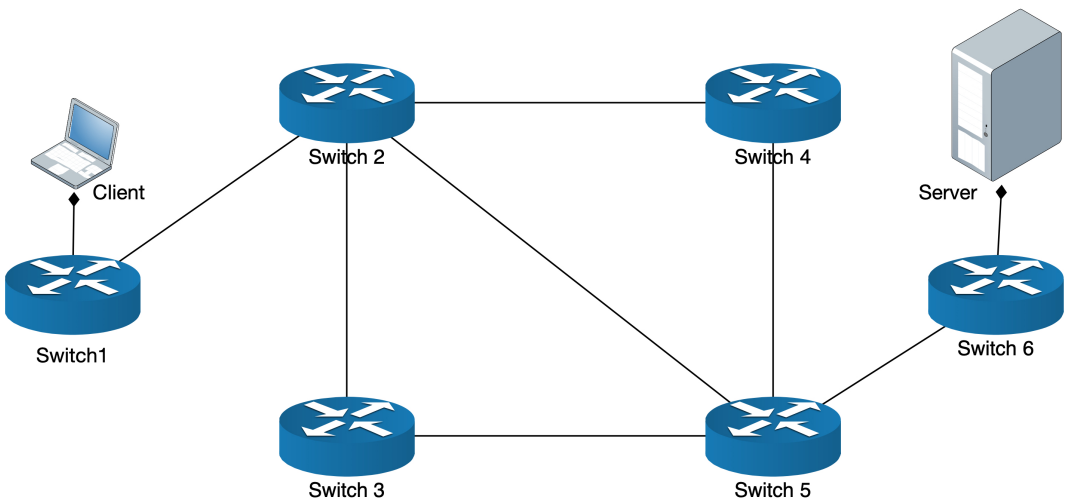


Figure 5.2. The topology without using a Linux Bridge.

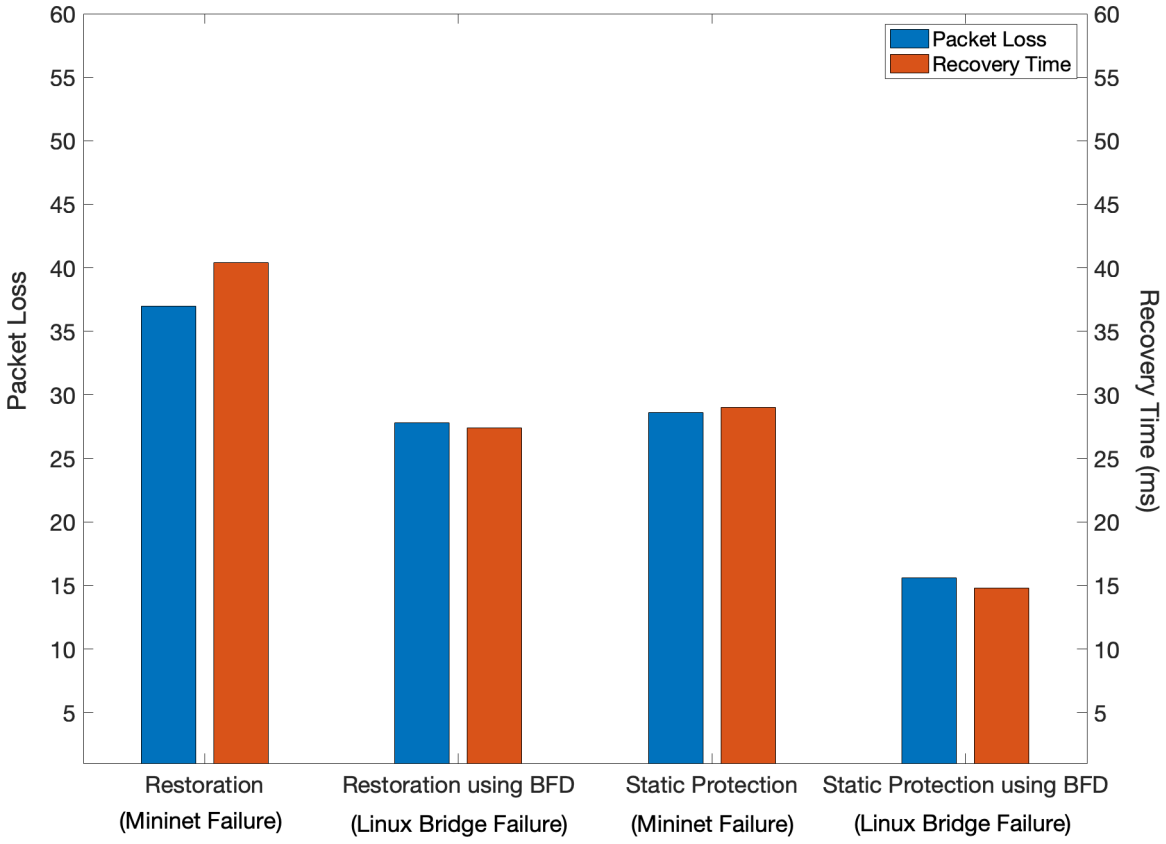


Figure 5.3. Comparison of packet loss and recovery time including restoration and static protection modules.

Switch 2 and 5 using only the Mininet command. We used a single server attached to Switch 6 and all clients were attached to Switch 1. Thus, each flow was routed on the same path as S1-S2-S5-S6. All experiments lasted 30 seconds in which the failure is triggered at the 15th second.

To evaluate the effect of the number of flows, we first compared the performance of the restoration and static protection approaches using the packet loss rate as the performance metric as shown in Figure 5.4. The result shows that, using the restoration approach, if the flow count passing through the faulty link increases, the packet loss rate also increases. On the other hand, if we use the static protection method, the packet loss rate would be stable while the number of flows are increased. Since each flow rule is processed by the controller sequentially, increasing number of flows bring about more packet loss. On the other hand, since all primary and secondary paths are predefined in the static protection approach, increasing number of flows does not affect the packet loss.

Apart from evaluating the packet loss rate of the affected flows, we also considered the control path traffic triggered by the failure. Since all operations and configurations in SDN are performed via the controller, the traffic on the control path is crucial. Therefore, the control path traffic should not be bursty/congested that may cause an interruption on the services deployed in the network. To this end, we compare the restoration and static protection approaches in terms of control path traffic considering the increasing number of flows. To measure the control path traffic, we applied the Wireshark tool at the network interface of the controller that connects to the data plane in Mininet.

The results shown in Figure 5.5 and Figure 5.6 present the tradeoff between the restoration and static protection approaches regarding the control path traffic. In the restoration approach, when the failure occurs at the 15th second, the control path traffic increases as shown in Figure 5.5(a), 5.5(c), 5.6(a), and 5.6(c). Therefore, if there is a critical job in the network, it would be interrupted for several seconds. Moreover, if the network includes thousands of flows managed by the controller, increasing number

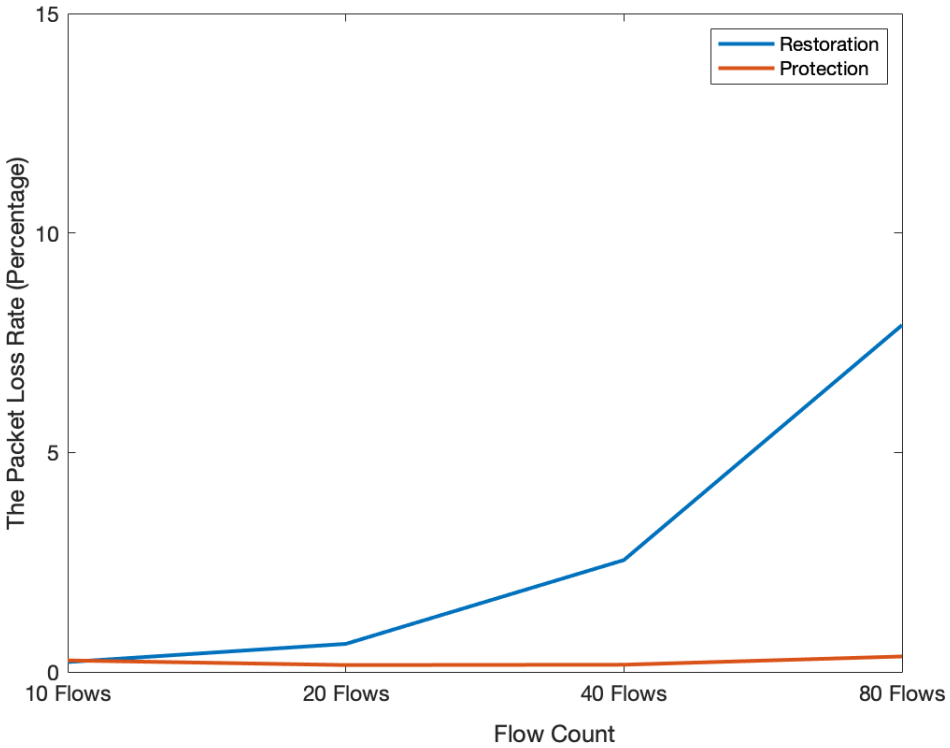
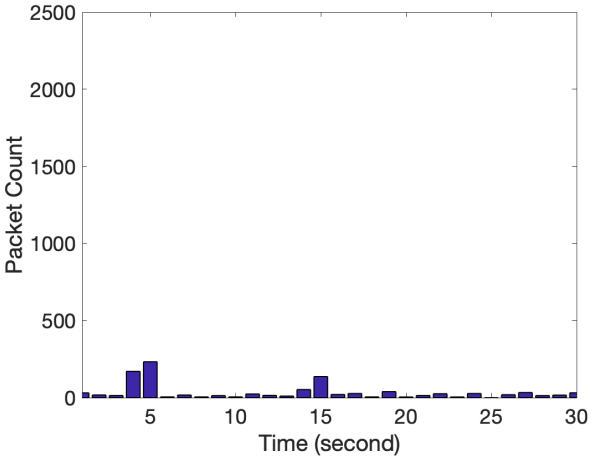


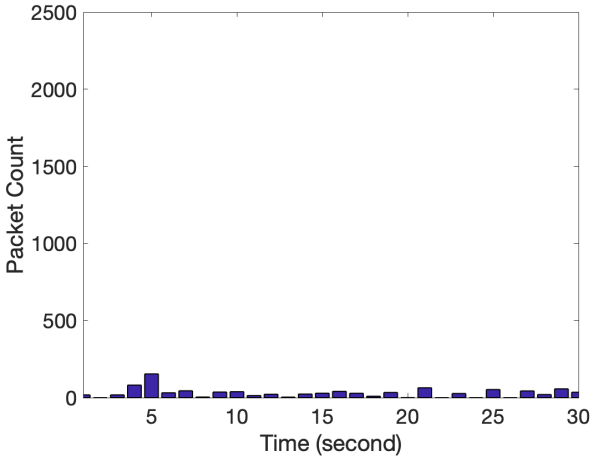
Figure 5.4. The packet loss rate based on the flow count and used fault tolerance method.

of affected flows may cause disastrous scenarios.

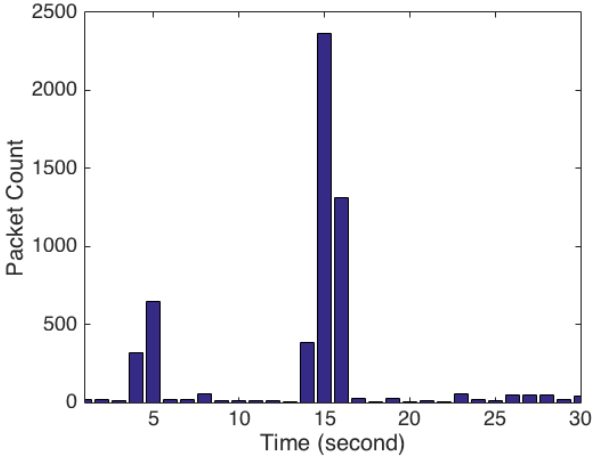
On the other hand, the static protection approach does not affect the control path traffic when the failure occurs as shown in Figures 5.5(b), 5.5(d), 5.6(b), and 5.6(d) since the alternative paths have already been configured in the switches in the case of a failure on the link. As a result, a network that includes thousands of flows in a datacenter may not be affected by a link failure in contrast to the restoration approach. However, since static protection approach requires to install alternative flow rules at the beginning of the communication, it causes bursty traffic on the control path in this case. Nonetheless, the moment of the bursty traffic on the control path may be configured based on the load of the network since the start of the communication can be observed and managed by the controller. Thus, we can conclude that the static protection approach performs better than the restoration approach in the case of multiple flows.



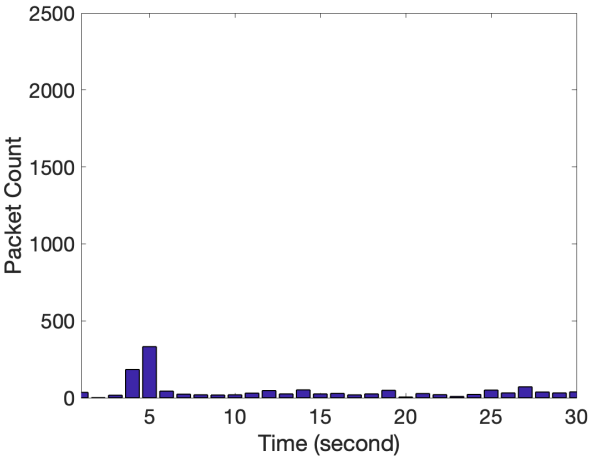
(a) Control path traffic for restoration with 10 flows



(b) Control path traffic for static protection with 10 flows

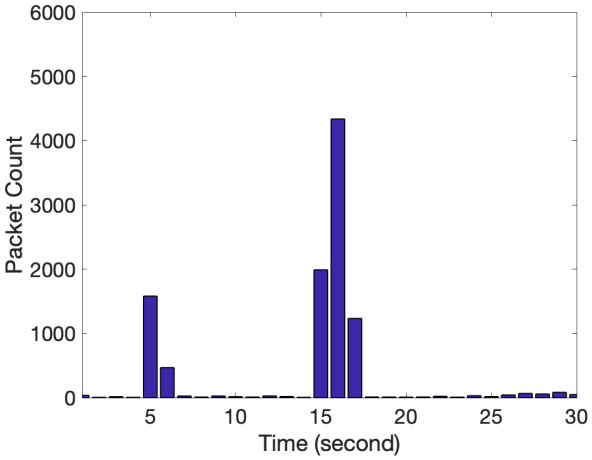


(c) Control path traffic for restoration with 20 flows

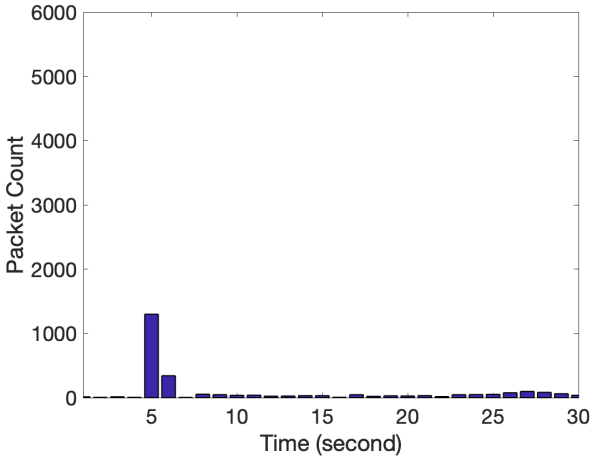


(d) Control path traffic for static protection with 20 flows

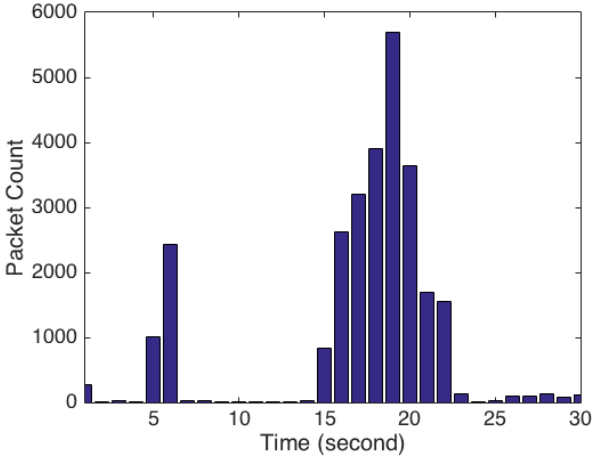
Figure 5.5. The number of packets of the control path traffic during the lifetime of the experiment.



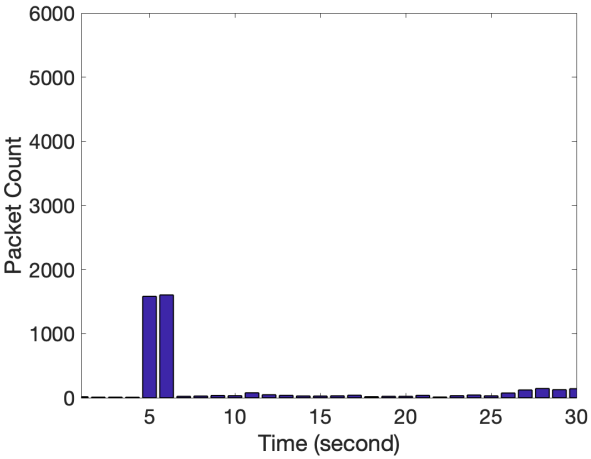
(a) Control path traffic for restoration with 40 flows



(b) Control path traffic for static protection with 40 flows



(c) Control path traffic for restoration with 80 flows



(d) Control path traffic for static protection with 80 flows

Figure 5.6. The number of packets of the control path traffic during the lifetime of the experiment.

5.3. Effect of the Size of Topologies

We used four different topologies shown in Figure 5.7, 5.8, 5.9, and 5.10 in order to evaluate the effect of different topology sizes on the packet loss in the case of a link failure. Since we would like to observe only the impact of topology sizes in this case, we used a single flow in our experiments for this evaluation. In our experiments, the primary path on each topology has the same hop count. However, considering the failure on the link between Switch 2 and 5, each topology has different complexity based on their number of alternative paths that are measured from Switch 2, the source node after the failure, to Switch 6, the destination node. Therefore, we named these topologies regarding their number of alternative paths as *TwoPathsTopology*, *FivePathsTopology*, *FourteenPathsTopology*, and *TwentyNinePathsTopology*. The duration of our experiments was 30 seconds in which the failure was triggered at the 15th second.

The results in Figure 5.11 show that the increasing complexity of the topologies affects the packet loss negatively when we use the restoration approach for the fault tolerance. Since it takes more calculation time to compute the secondary path for the increasing number of alternative paths, consulting the controller in the restoration approach causes more packet loss for each increasing complexity. On the other hand, the packet loss is not affected by the size of the topologies when we use the static protection approach since the alternative paths have already been calculated and installed for relevant switches. As a result, the static protection approach performs better than the restoration approach in this case.

5.4. Evaluation of Linux Bridge and BFD Intervals

To evaluate the effect of the Linux bridge command in order to observe the broken link scenario in Mininet, we first compared the transmission patterns of the link failures carried out by Mininet command and Linux bridge command, which consists of shutting down a port of Linux bridge, respectively. We applied the restoration and static protection approaches for the evaluation. We used the topologies shown

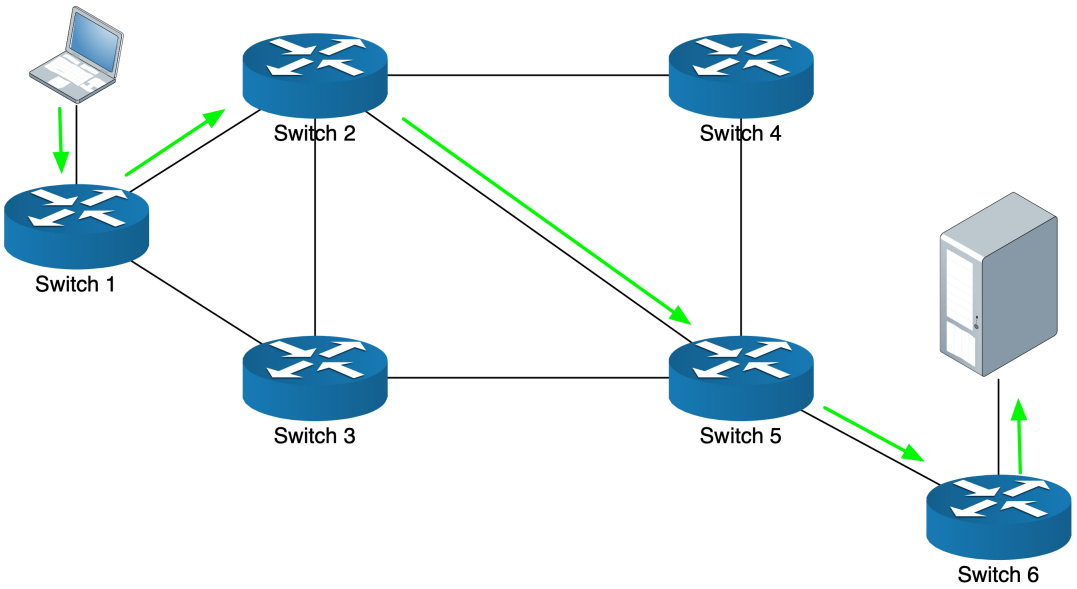


Figure 5.7. *TwoPathsTopology* that has two different alternative paths after the failure at the link between Switch 2 and 5.

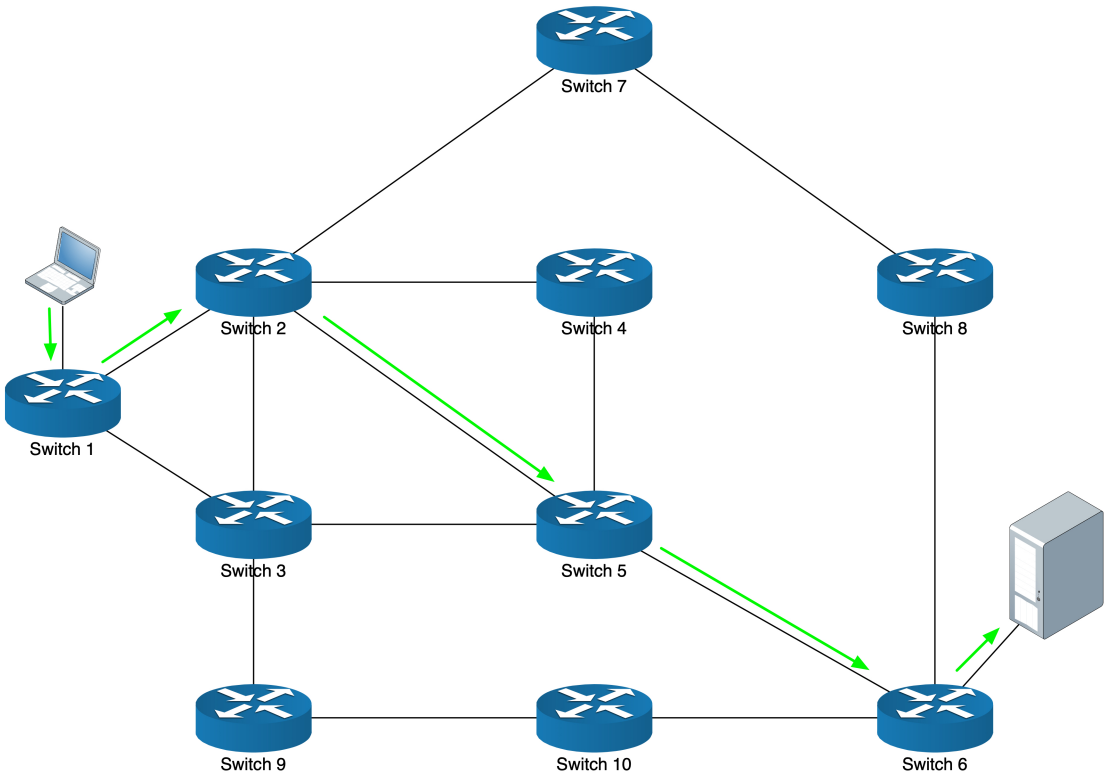


Figure 5.8. *FivePathsTopology* that has five different alternative paths after the failure at the link between Switch 2 and 5.

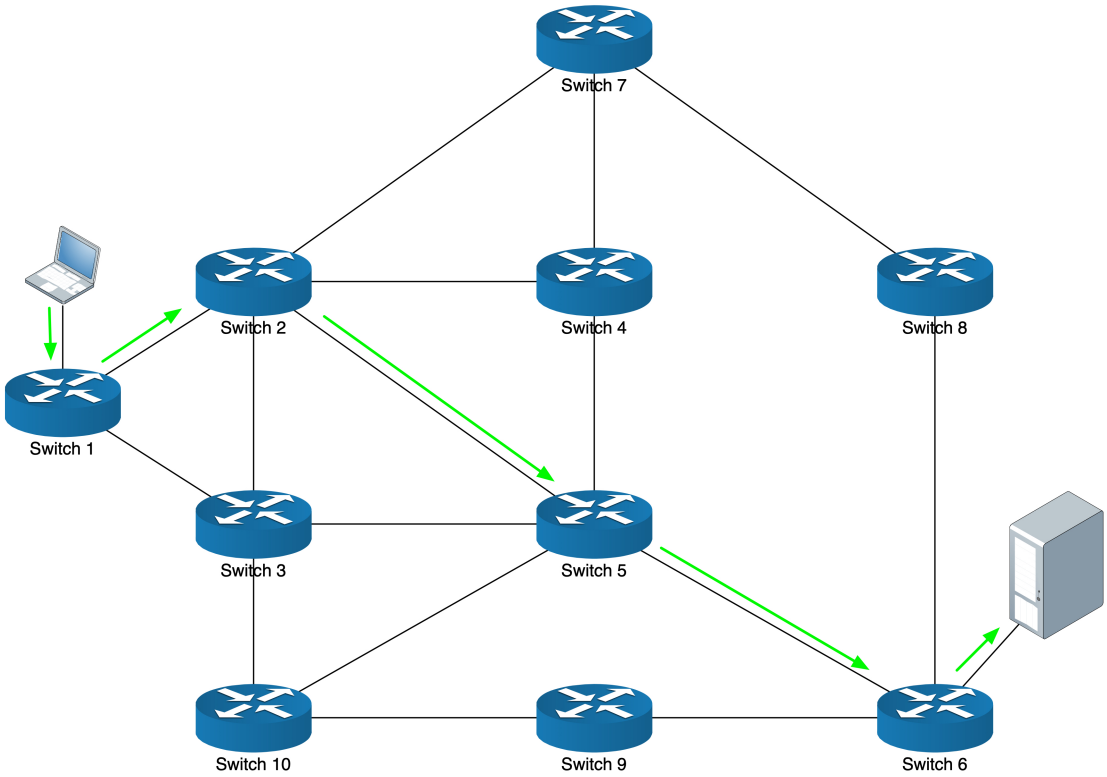


Figure 5.9. *FourteenPathsTopology* that has 14 different alternative paths after the failure at the link between Switch 2 and 5.

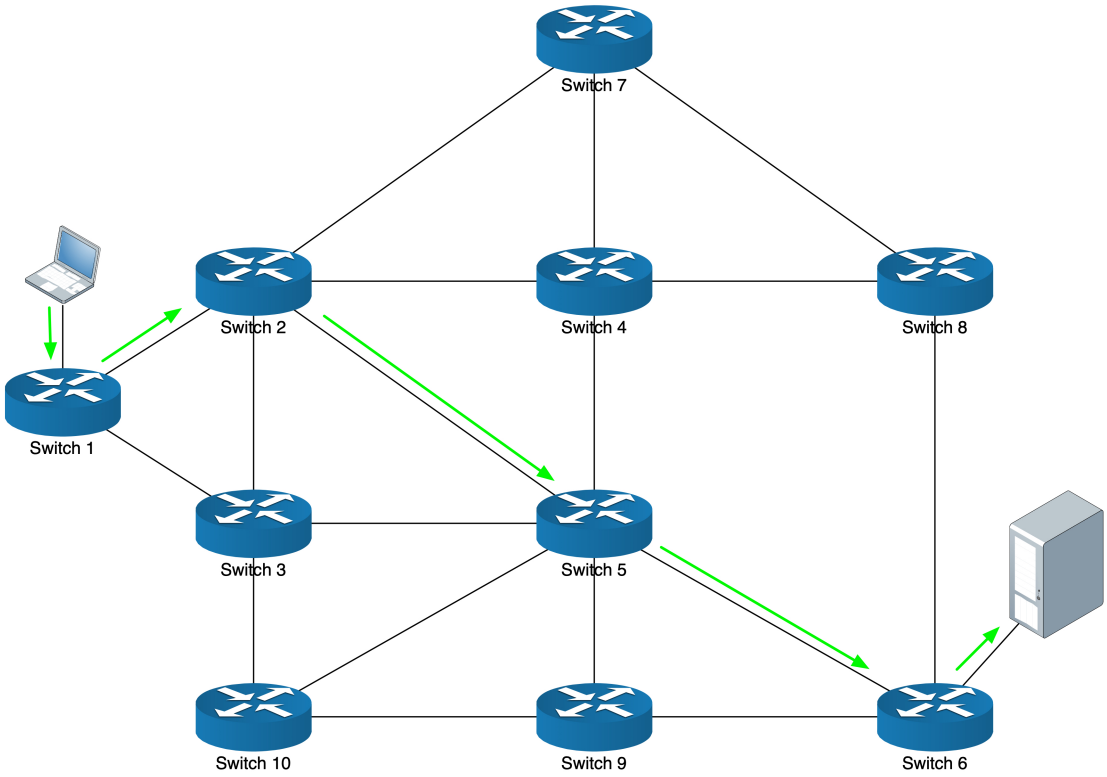


Figure 5.10. *TwentyNinePathsTopology* that has 29 different alternative paths after the failure at the link between Switch 2 and 5.

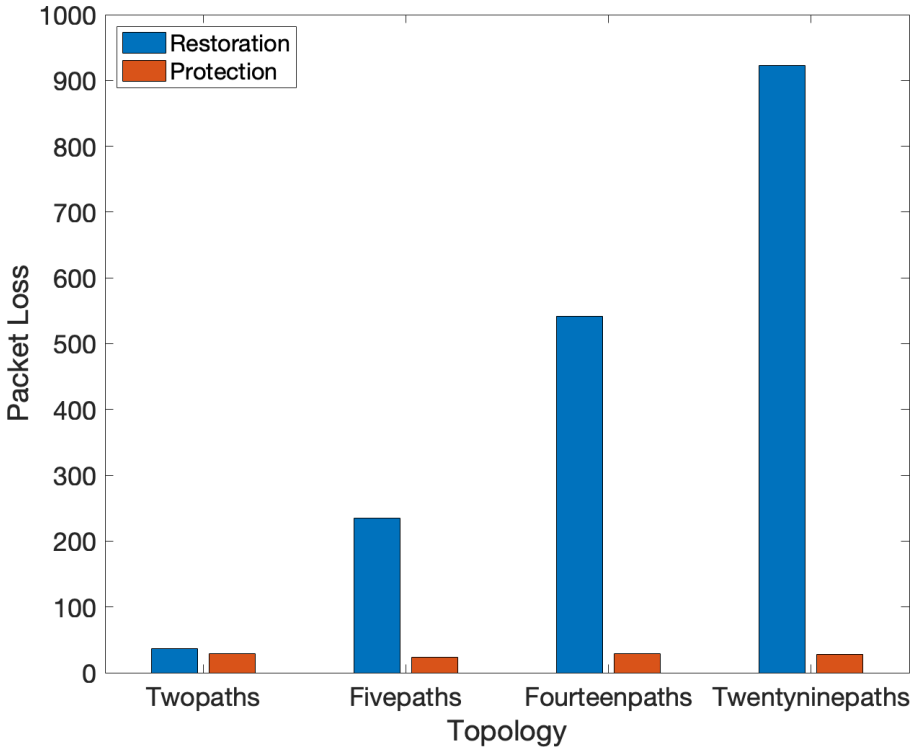
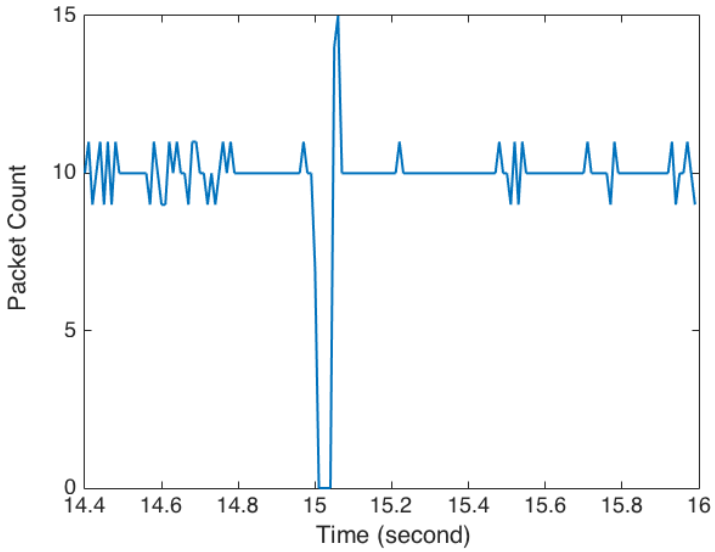


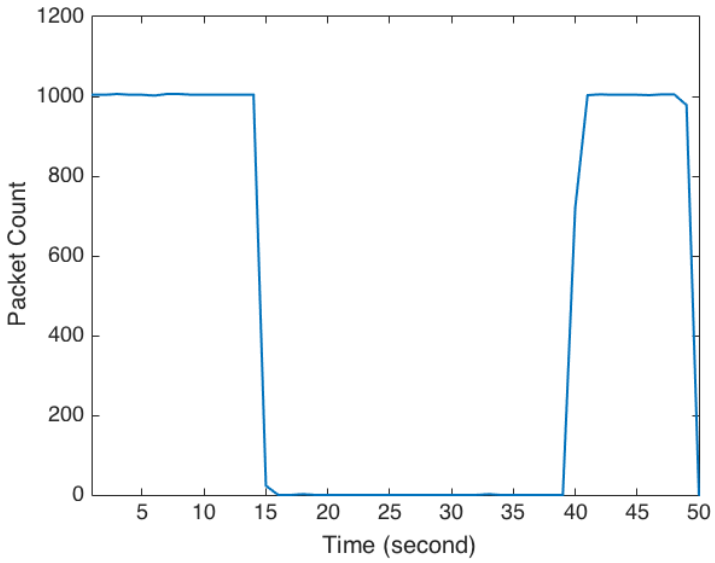
Figure 5.11. The packet loss based on the different topologies.

in Figure 5.1 and 5.2 for the Mininet-based failure and Linux bridge based failure, respectively. In the experiments, we used a single flow to observe the Linux bridge effect independently. The duration of each experiment was 50 seconds in which the link failure was created at the 15th second.

Considering the link failure carried out by the Mininet command, Figure 5.12(a) shows the pattern of the transmission for the restoration approach based on the packet count. This pattern also shows the recovery time of the restoration approach whose mean is 40ms as shown in Figure 5.3. On the other hand, if we shut down one of the ports of the Linux bridge to create the link failure, the failure notification event cannot be created immediately due to ports are active and therefore the controller cannot be notified. Thus, in this case, the controller deduces the link failure in the data plane via LLDP packets. Since LLDP packets are sent by the controller periodically for the topology discovery update in seconds to prevent the burden on the controller, the failure detection time is much higher than the other methods. Moreover, the failure detection time using LLDP updates in the Floodlight controller is twice of the LLDP update time, which is 12 seconds in our module. Thus, the pattern of the throughput



(a) Recovery time is 30-50ms for restoration module using Mininet command.



(b) Recovery time is in seconds for restoration module using Linux bridge for failure.

Figure 5.12. The demonstrative examples of failure recovery for different scenarios.

in this case is as shown in Figure 5.12(b). However, if we use the BFD protocol between Switches 2 and 5 using T_i as 5 ms, the mean of the recovery time would become 27 ms that is independent of the failure creation approaches including Linux bridge and Mininet commands as shown in Figure 5.13.

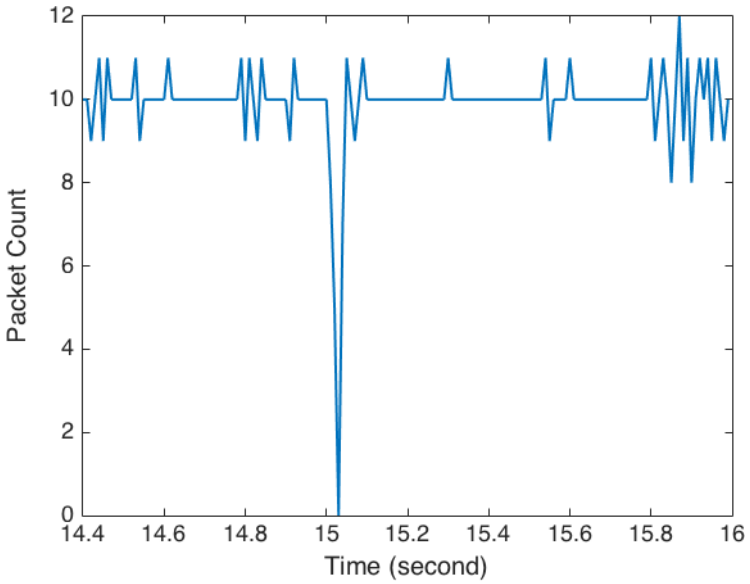


Figure 5.13. BFD significantly reduces recovery time independent from failure methods.

We applied the same approaches above for the evaluation of the static protection module. Since this module solves the problem in the data plane using OpenFlow Fast Failover groups, the recovery time for the link failure created by the Mininet command is smaller than the restoration module with the mean value of 28ms. However, on the other hand, if we shut down the port of the Linux bridge for the failure, the transmission of the packets halt as shown in Figure 5.14. Since the relevant ports of Switch 2 and 5 are not affected by the failure in this case, OpenFlow FF groups continue to operate even though there is a fault on the link between them. Thus, packets are sent to the ports that are assumed as working and then the traffic stop. However, if we run the BFD protocol on that link, the transmission continues flawlessly since Open vSwitch checks the liveliness of the ports based on the value of T_i. Therefore, using BFD, since the controller cannot be involved, the mean of the recovery time is 15ms in this module.

Since we have observed through our experiments that BFD is crucial for a fault tolerant system, we also evaluated the effect of different T_i values on the recovery. We considered 15ms, 45ms, and 90ms failure detection times regarding Equation 2.2 and measured the packet loss. The result shown in Figure 5.15 presents that when T_i increases, the packet loss also increases since the duration of the failure detection is prolonged.

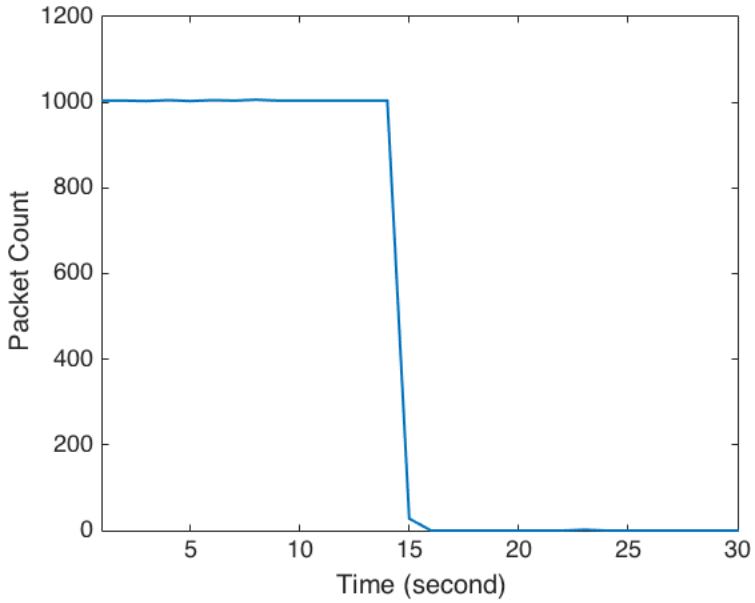


Figure 5.14. The transmission of packets stop when we use Linux bridge for the failure for static protection module.

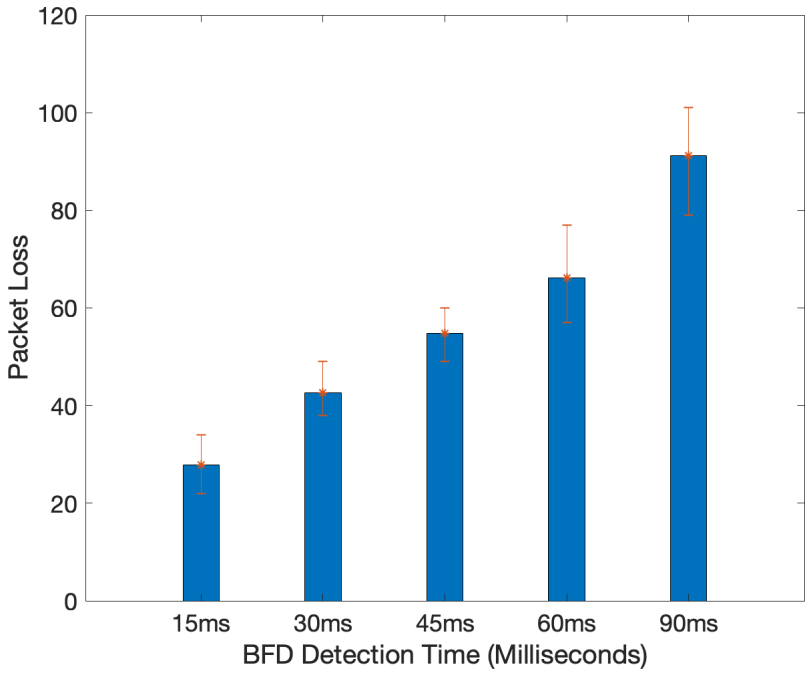


Figure 5.15. Packet Loss based on the BFD message interval.

5.5. Evaluation of the Quality of the Alternative Paths

For a proper assessment of the importance of the QoAP, we compare our dynamic protection module with the static protection module. In the test scenario, we create a heavy traffic on the secondary path before the link failure. Since affected flows are

sent to the secondary path without considering the network conditions in the static protection, this comparison presents the benefits of considering the QoAP.

In our experiments each of which lasts 50 seconds, we used the topology shown in Figure 5.2. In the topology, the primary path is S1-S2-S5-S6 while the secondary path is S1-S2-S3-S5-S6. We first created the traffic that causes 100% load on the link between Switch 2 and 3 at the 10th second. Afterwards, at the 26th second, we applied the Mininet command to create the link failure on the link between Switch 2 and 5.

The result shown in Figure 5.16 depicts that the throughput of the transmission significantly reduces when the QoAP is not considered. Moreover, using dynamic protection, the traffic is not affected by the conditions of the original secondary path after the link failure since all affected flows have been directed into a different route.

On the other hand, evaluation of the interval parameter that is used to calculate the QoAPs for the given topology is crucial for the performance of the system. Therefore, we compared the performance of 2-sec, 4-sec, 7-sec, and 10-sec intervals considering the packet loss. Moreover, we also compared these results with the performance of the static protection. The results shown in Figure 5.17 presents that the performance of our dynamic protection approach outperformed the static protection. Since QoAPs are not considered in the static protection, packet losses are higher than the dynamic protection. Besides, the results also show that if the interval in dynamic protection decreases, the packet loss reduces since the system uses the most updated information of the network. However, lower interval causes higher cost for the system since the statistics of the all living ports in the network must be collected within the given interval via LLDP packets. Thus, the interval must be optimized considering the controller traffic and the applications that run on the network.

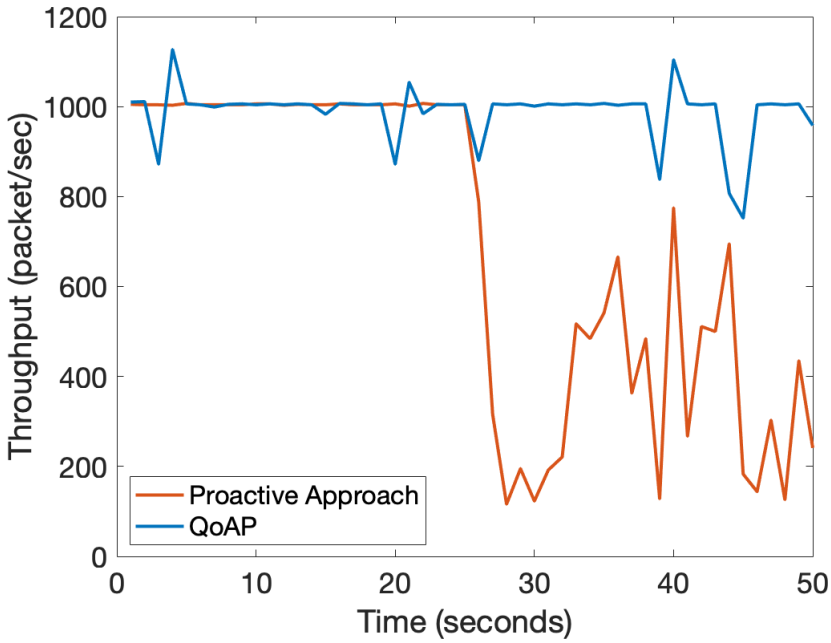


Figure 5.16. Comparison of the QoAP and traditional protection approach based on packet loss for each QoAP interval.

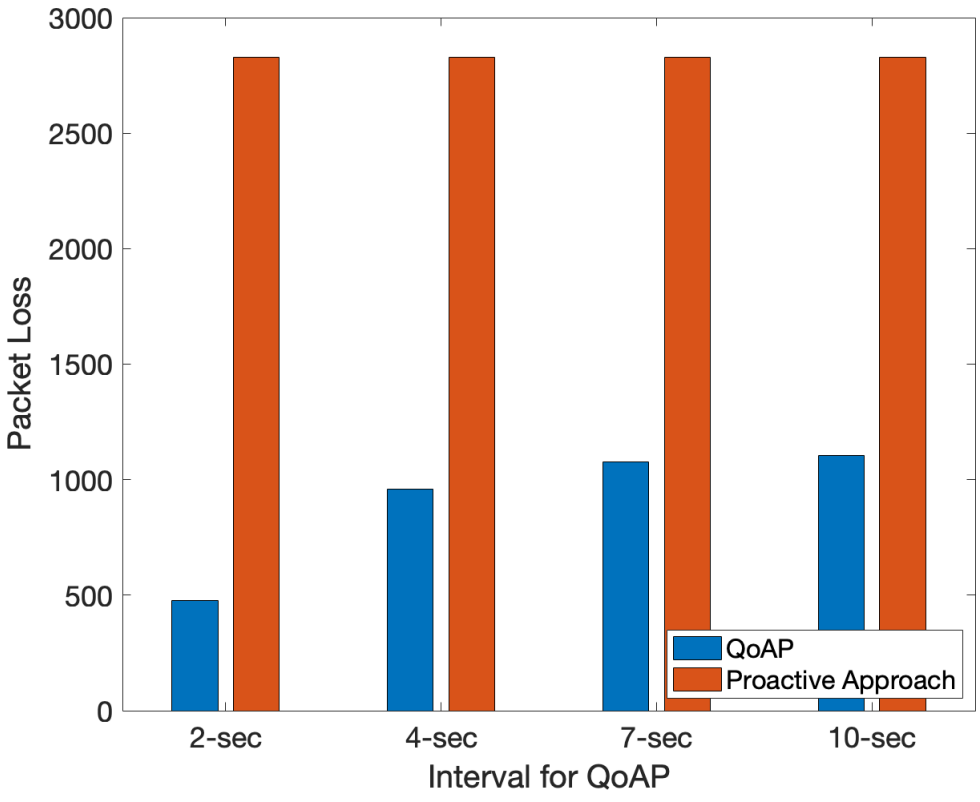


Figure 5.17. Differences between static and dynamic protection modules considering quality of alternative paths.

6. PERFORMANCE EVALUATION OF APPLICATION-BASED METRICS

Evaluating the change of the application-based metrics when the failure happens in the network is crucial as well as assessing the network-based parameters since end-users are affected in this case. Therefore, our scenarios for this evaluation is similar to the network-based assessment considering the performance of the restoration, static protection and dynamic protection modules in the case of a link failure. In the experiments, we used the topology shown in Figure 5.1 including one client and one server. The client is connected to Switch 1 and the server is connected to Switch 6. We used a short movie namely Big Buck Bunny with 1080p resolution for the video streaming. The length of the movie is 600 seconds and the link failure was carried out at the 300th second for each experiment. Moreover, we used 1-second and 10 seconds segment sizes in order to assess the effect of the segment size. We investigated the change of the video quality value and buffer level as the application metrics considering QoE. To measure the video quality value, we used Equation 2.5 while the buffer level information was provided by DASH API.

6.1. Restoration

When the restoration module is used, the video quality and buffer level is not affected by the failure created by the Mininet command for both segment sizes as shown in Figure 6.1(a) and 6.1(b). Since video streaming is based on HTTP and the buffer can hold the fragments of the video for seconds, a failure, which is recovered within milliseconds, does not affect the quality.

On the other hand, when we use the Linux bridge to create the link failure, the buffer level and therefore the video quality is affected for seconds since the recovery time is based on the LLDP update interval period in this case. As a result, QoE of the user reduces for seconds as shown in Figure 6.2(a) and 6.2(b). Another important

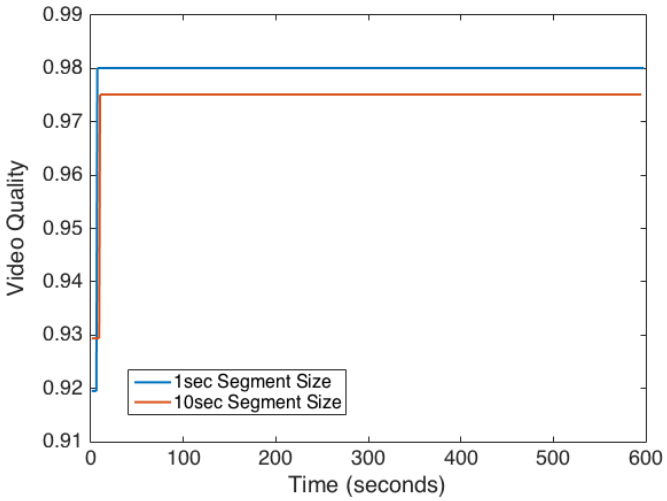
result is that the failure impacts the video consisting of 10 seconds segment size more than the 1-second segment size. The main reason is that filling a 10 seconds long segment after the failure requires longer time than that of the 1-second segment size. Thus, the video cannot be continued to play properly because of the DASH concept which requires a full segment for streaming.

6.2. Static Protection

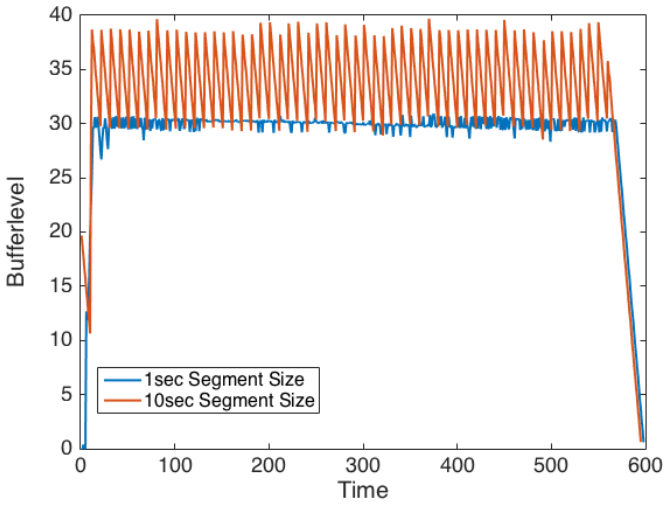
Using the static protection module, the buffer level and video quality are not affected by the failure created by the Mininet command as well as the restoration module. However, if we use the Linux bridge to create the failure, the transmission of the packets halt as in the evaluation of QoS using iPerf. Thus, the video continues a few seconds after the failure until the buffer in the client is depleted. On the other hand, when we activate the BFD protocol on the link between Switch 2 and 5, the buffer level and the video quality was not affected by the failure as shown in Figure 6.1(a) and 6.1(b).

6.3. Dynamic Protection

To evaluate the performance of the dynamic protection for the video application, we applied a similar scenario carried out in Section 5.5. To this end, we created a heavy traffic on the secondary path at the 100th second before the link failure. Afterwards, the link failure was created on the primary path, between Switch 2 and 5, using the Mininet command at the 300th second. We did not use the Linux bridge command for the failure in this case since our main consideration was to evaluate the quality of the communication after the failure rather than evaluating the recovery time. The result shown in Figure 6.3(a) and 6.3(b) points out that the QoE significantly reduces after the failure if we use the static protection in this scenario. On the other hand, if we use our dynamic protection module, QoE is not affected by the failure. Thus, it is clearly seen that considering the quality of alternative paths is also crucial for applications in a fault tolerant system as well as the recovery time.

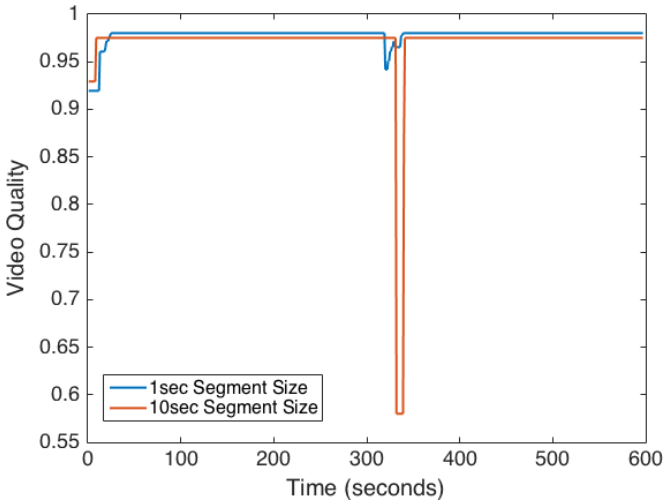


(a) Video quality value when we use Mininet command for the failure.

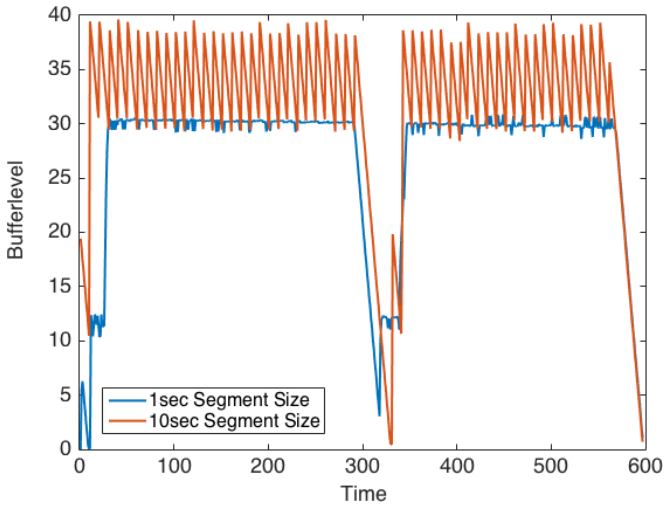


(b) Buffer level of the video when we used Mininet command for the failure.

Figure 6.1. The effect of failure recovery on the video quality and buffer level of video streaming using restoration module with Mininet-based failure.

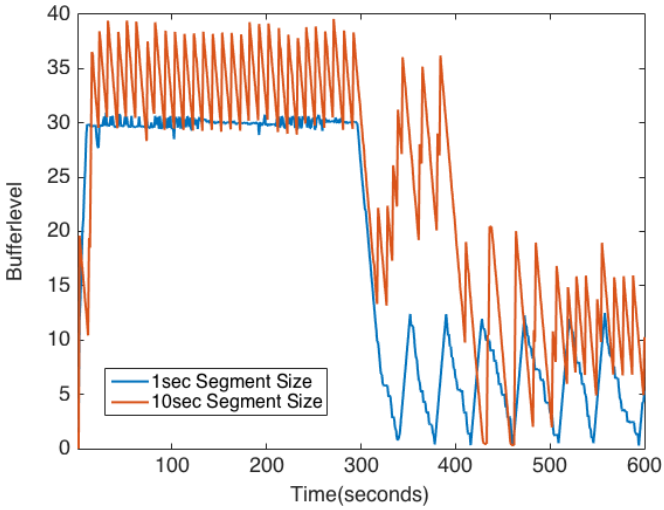


(a) Video quality value is affected when we use Linux bridge for the failure.

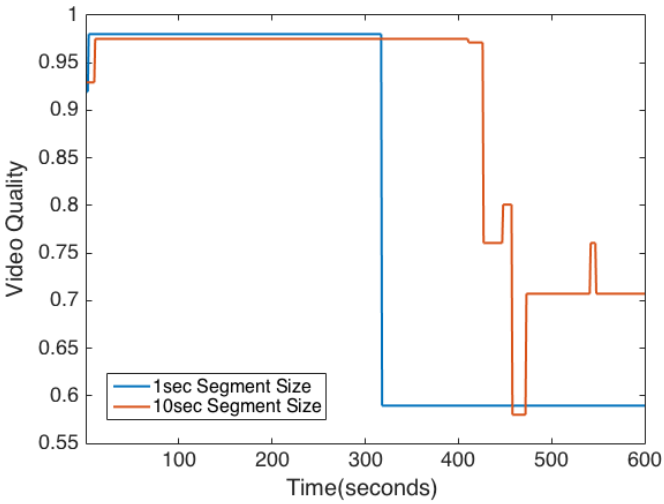


(b) Buffer level is affected when we use Linux bridge for the failure.

Figure 6.2. The effect of failure recovery on the video quality and buffer level of video streaming using restoration module with Linux bridge-based failure.



(a) Buffer level without considering QoAP.



(b) Video quality value without considering QoAP.

Figure 6.3. QoE is significantly affected if the quality of the alternative paths is not considered for fault tolerance.

7. PERFORMANCE EVALUATION OF BFD-BASED CONGESTION DETECTION ON QoE

We investigated three factors in our experiments: the impact of the BFD interval, the traffic load, and the video segment size on the QoE parameters. For each experiment, we used Mininet for SDN emulation deploying Open vSwitch for switches since it supports both BFD and OpenFlow protocols. On the other hand, we used DASH.js for the video clients. To generate additional traffic and thus cause congestion, we run the iPerf tool on Mininet.

For each experiment, the topology shown in Figure 7.1 was used. We limited the capacity of all links as 50Mbps by deploying five DASH clients connecting to Switch 1 and one DASH server attaching to Switch 6. We ensured that all DASH clients obtain the video segments via the same route as indicated in Figure 7.1 as green arrows. On the other hand, four iPerf clients were connected to Switch 2 to cause congestion by sending their packets to the iPerf server which is located at Switch 5. Moreover, Big Buck Bunny short movie, the duration of which is 10 minutes, is used for streaming the 1080p resolution video. After the streaming was started for each DASH client, T1, T2, T3, and T4 iPerf clients began to send their packets at 50th, 80th, and 110th second respectively to cause congestion. These iPerf clients generated the same amount of UDP traffic and induced congestion on the link between Switch 2 and Switch 5.

We used 1 second and 10 seconds segments to evaluate the effect of the segment size. To observe the impact of the traffic load, we generated 40 Mbps (80% Load), 45 Mbps (90% Load), and 49 Mbps (98% Load) traffic using only the iPerf clients. Finally, to evaluate the influence of the BFD intervals in our system, we used T_i as 100ms and 1000ms respectively by taking M value as two. Thus, the T_d values were 300ms and 3000ms respectively. Moreover, we compared these results with the non-BFD case.

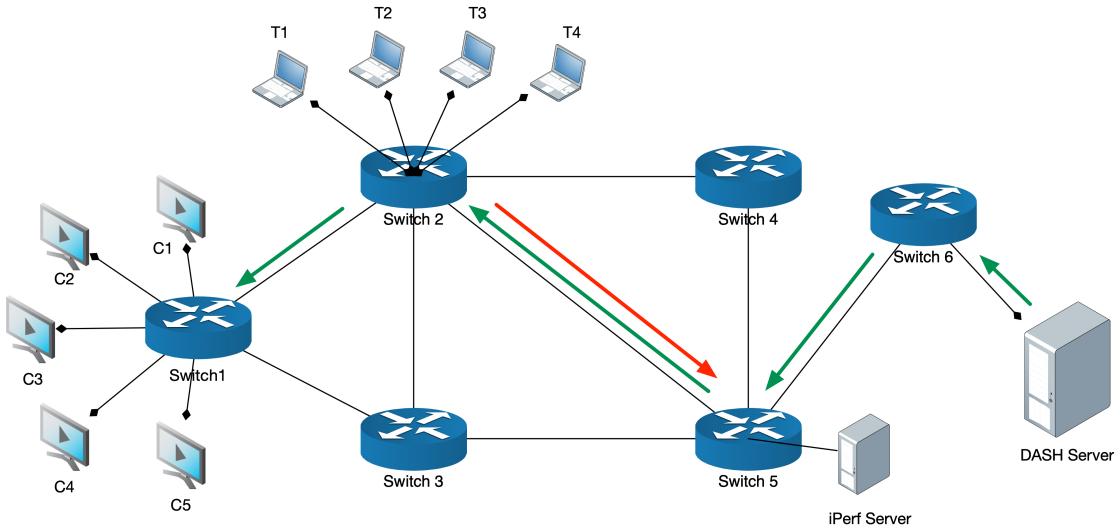
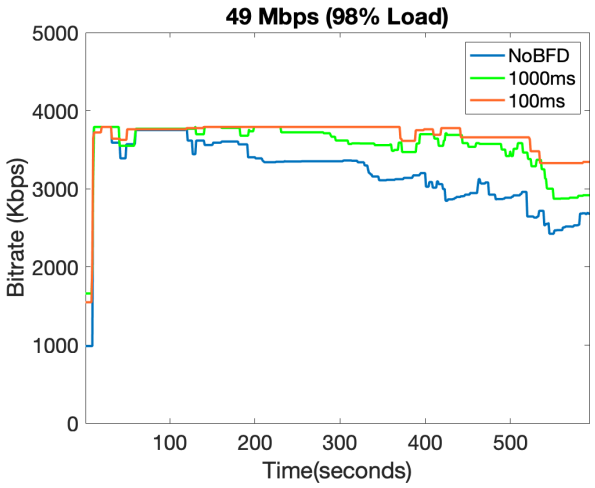


Figure 7.1. The topology and video traffic route

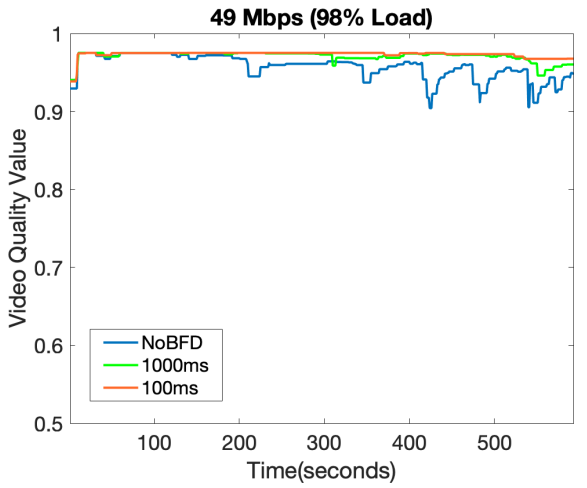
Consequently, we conducted our experiments for 18 different cases considering those three parameters. For each case, we repeated experiments 6 times. Since each test lasts 10-11 minutes, the duration of our experiments was 18 hours. To evaluate the results, we analyzed four QoE parameters including the bitrate, quality value, latency and number of quality switches from each client and calculated the average values for each case.

7.1. Effect of Segment Size

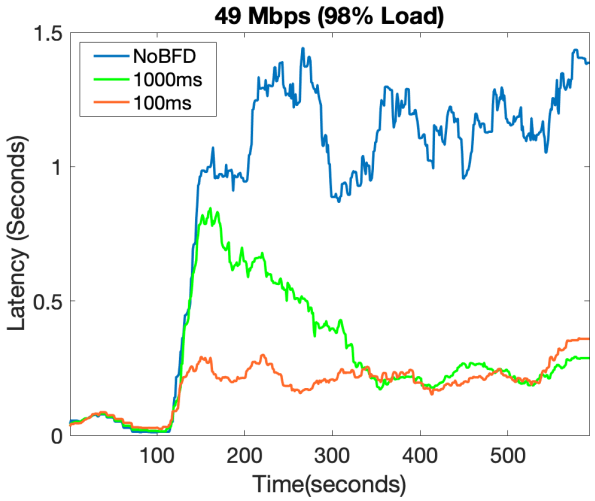
Our experiments showed that streaming with the big segment size is more stable than the small segment size considering the congestion conditions on the link. Experimental results for 49 Mbps (98% Load) traffic on the congested link given in Figure 7.2 and Figure 7.3 represent that fluctuations and change of the QoE parameters including the average bitrate, video quality, latency, and number of quality switches between representations are higher in 1-sec segment size compared with 10-sec segment size. This pattern is the same for 45 Mbps and 40 Mbps traffic loads. Since a small segment size needs more HTTP requests to transmit video segments, it is affected by the network conditions more than the big segment sizes.



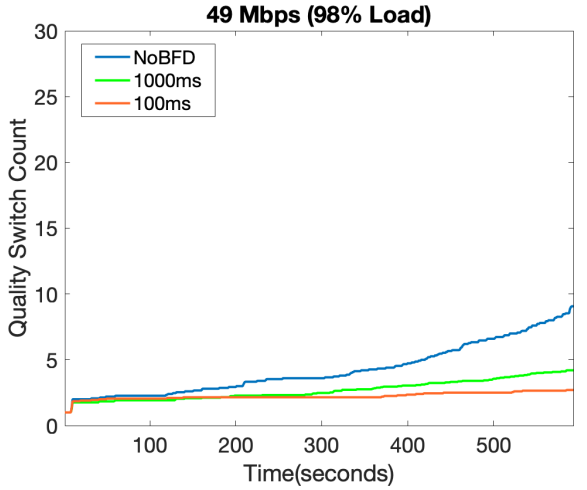
(a) Average bitrate for 10-sec segment size



(b) Average video quality for 10-sec segment size

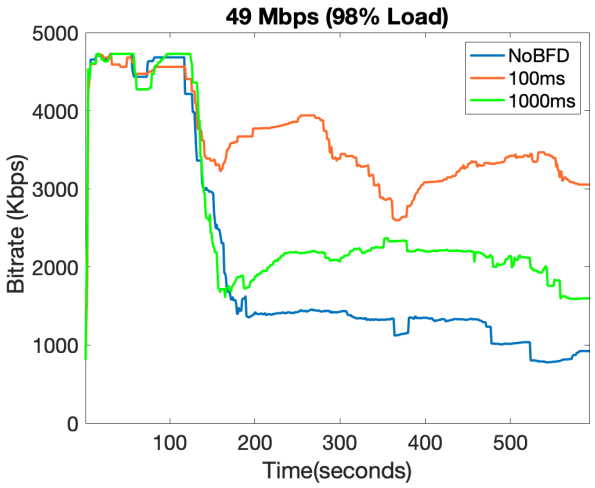


(c) Average latency for 10-sec segment size

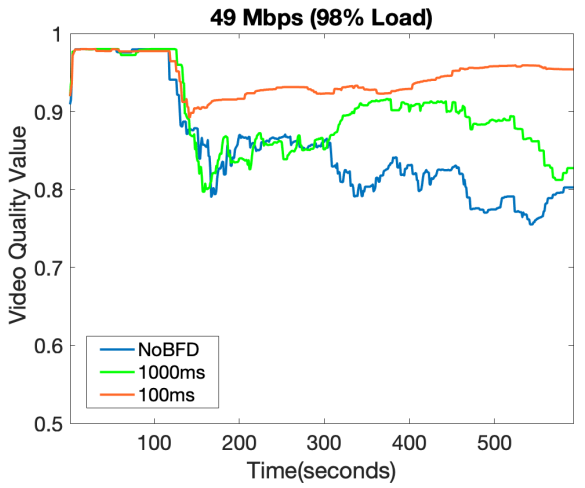


(d) Number of quality switches for 10-sec segments

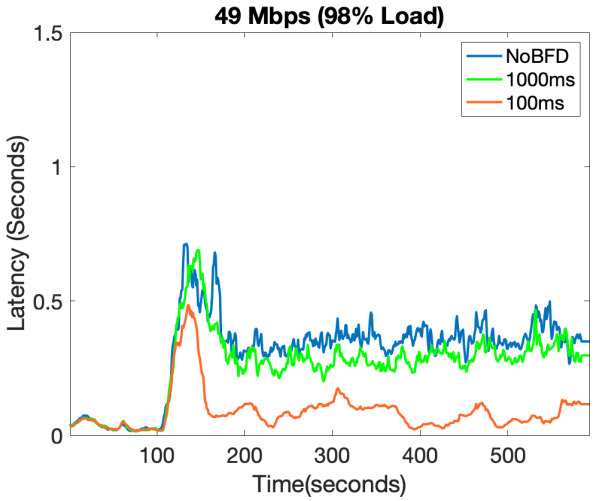
Figure 7.2. The change of QoE parameters for the congested link with 98% load.



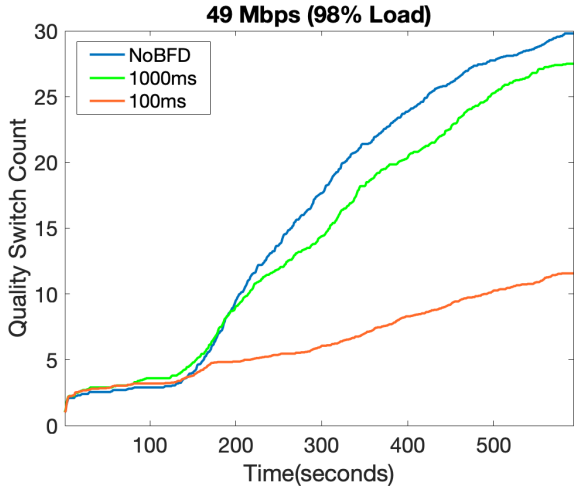
(a) Average bitrate for 1-sec segment size



(b) Average video quality for 1-sec segment size



(c) Average latency for 1-sec segment size



(d) Number of quality switches for 1-sec segment

Figure 7.3. The change of QoE parameters for the congested link with 98% load.

7.2. The Effect of Traffic Load

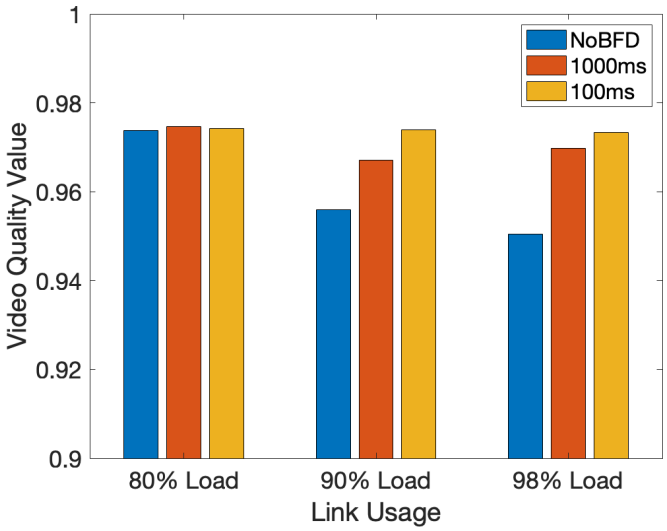
To evaluate the impact of the traffic load, we measured the average video quality of clients based on SSIM and the number of quality switches including all cases in our experiments. Our results demonstrated that when the traffic load increases, the video quality decreases considering both segment sizes with the non-BFD case as shown in Figure 7.4(a) and 7.4(b). For each traffic load, the video quality with 10-sec segment size is better than the 1-sec segment size for the non-BFD case due to its buffer capacity. On the other hand, if we use BFD for the congestion detection, the video quality is improved.

Considering the non-BFD case for the 80% and 90% traffic loads, the average video quality with 1-sec segment size is not so affected while the average video quality with 10-sec segment size is decreased. However, for the 98% traffic load, the video quality is poor when we used 1-sec segment size while it is acceptable for 10-sec segment size.

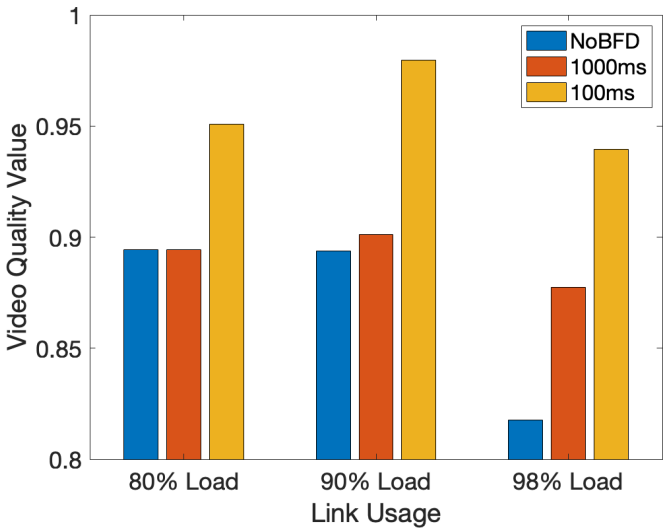
On the other hand, the effect of the traffic load on the number of quality switches is shown in Figure 7.5(a) and 7.5(b). The results show that the quality switch count between representations for the 10-sec segment size is the lowest for 80% load and highest for the 90% load when the BFD is not used. This is originated by the fact that 80% load is not heavy for the 10-sec segment size so that the quality change is low, while the quality is affected by the 90% load that cause quality switches. Moreover, since 98% load is the most influential for the quality, the quality cannot fluctuate so that the switch count is not higher than the case of 90% load. Besides, considering the 1-sec segment size, the count of switches between representations decrease for higher traffic loads since they cause worse video quality respectively.

7.3. The Effect of BFD Intervals

Our results showed that the impact of BFD is crucial in the case of congestion. In Figures 7.2 and 7.3, the results show that using BFD fixes the poor outputs of each

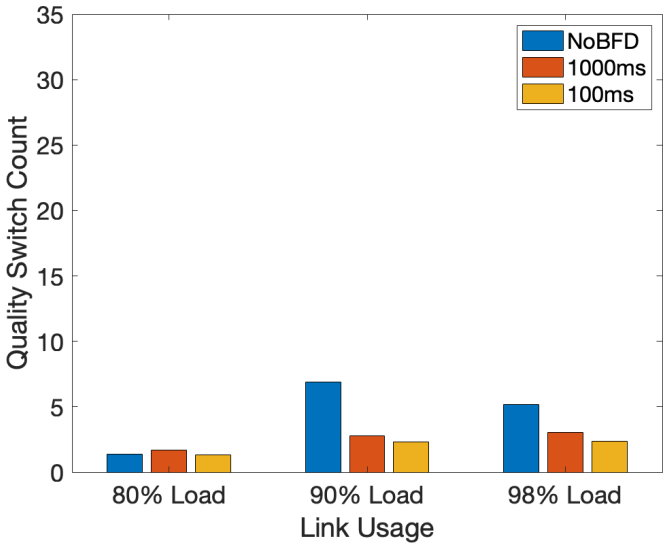


(a) The impact of the traffic load on the average video quality using 10-sec segment size

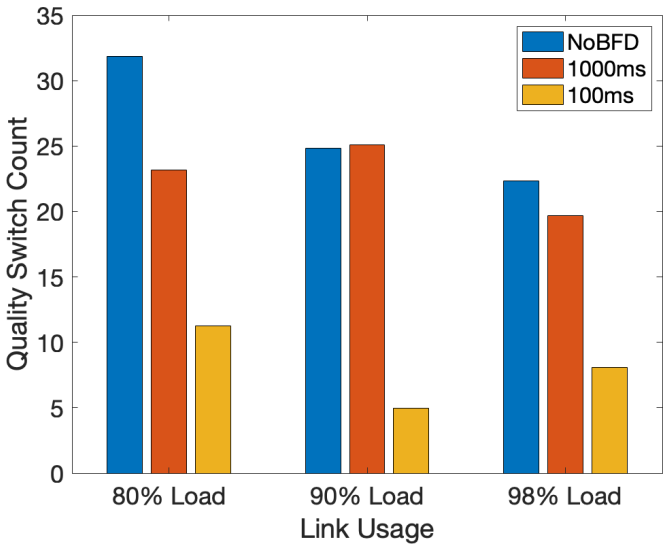


(b) The impact of the traffic load on the average video quality using 1-sec segment size

Figure 7.4. The impact of the traffic load on the average video quality with respect to different segment sizes



(a) The impact of the traffic load on the average quality switch count using 10-sec segment size



(b) The impact of the traffic load on the average quality switch count using 1-sec segment size

Figure 7.5. The impact of the traffic load on the quality switch count with respect to different segment sizes.

QoE parameters after the 150th second at which the traffic load starts to influence video streaming. Moreover, it is clearly observable that T_i with 100ms outperformed T_i with 1000ms regarding QoE parameters since its sensitivity is more delicate so that it detects the congestion earlier. Apart from the 49 Mbps shown in Figures 7.2 and 7.3, this pattern is also the same for other traffic loads including 45 Mbps (90% load) and 40 Mbps (80% load).

On the other hand, the effect of BFD is also noticeable in Figures 7.4(a) and 7.4(b) considering the video quality based on the traffic load. For 10-sec segment size with 80% traffic load, the effect of BFD interval is limited since the traffic could not cause congestion that induces to prevent BFD control packets considering their message interval, T_i. However, for 1-sec segment size with 80% traffic load, T_i with 100ms is affected by the traffic so that the quality is improved. However, considering the 90% and 98% traffic loads, T_i with 1000ms is also affected by the congestion. Moreover, the number of quality switches between representations is reduced when we use BFD as shown in Figure 7.5(a) and 7.5(b). The number of quality switches is the lowest for the T_i with 100ms regarding 10-sec and 1-sec segment sizes.

8. CONCLUSIONS

In this study, we investigated the fault tolerance in the SDN data plane considering the network-based parameters including the recovery time, packet loss, and throughput for QoS, and application-based parameters including the video quality value based on SSIM, and buffer level for QoE. Since network-based parameters for fault tolerance in the SDN data plane were studied before, we focused on two important issues that were not examined before. First, we focused on the quality of alternative paths considering network-based parameters. Even though some studies mentioned the importance of the QoAPs, there is no study that implements this concept in the literature like our module, namely the dynamic protection. Based on the results of our experiments, we clearly state that considering QoAP for fault tolerance is crucial as well as the recovery time. Second, we focused on the creation of the failure in a realistic manner which is not considered in the literature. Studies using an emulation environment such as Mininet run a special command to create link failures which also destroys the ports of the switches. Therefore, all of the other studies evaluated their results based on the Loss of Signal failure detection method which generally occurs for the port failures rather than the link failures. As a more realistic alternative, we used the Linux bridge, which is placed between the two switches and completely transparent to the controller, to evaluate a real-world case and obtain realistic results in Mininet. Hence, we created a more realistic comparison between the two cases. Our experiments showed that link failures carried out by the Mininet command are recovered within milliseconds while link failures created by shutting down one of the ports of the Linux bridge are recovered in seconds if BFD is not used. However, if BFD is used in the experiments using the Linux bridge and Mininet, the failure may be recovered faster based on the message transmit interval of the BFD protocol.

We also studied the application-based parameters using DASH for fault tolerance. Since studies that focus on SDN data plane for fault tolerance considered only the network-based metrics, this is the first study that investigates how application and user experience are affected by the failure and the failure recovery mechanisms. To explore

this, we applied the same scenarios and modules used for network-based parameters in a scenario with video streaming clients. To evaluate QoE, we used the video quality value based on SSIM and the buffer level in the client. The results of the experiments showed that the QoE appears to be not affected if the failure is created by the Mininet command since it is recovered within milliseconds. On the other hand, if the link failure is created by using the Linux bridge and BFD is not used, the video and therefore the QoE is affected for seconds. Moreover, we showed that if QoAPs is not considered for the recovery, the recovery time would not be important since the QoE significantly reduces because of the network conditions that affect the selected path.

On the other hand, we considered the congestion case applying the BFD protocol, which is originally designed to detect failures between network nodes, in order to detect the congestion on the path through which the video flows passing. We investigated the effect of the video segment size, traffic load, and BFD intervals on several QoE parameters that reflect the subjective opinion of the users. We used 1 second and 10 seconds long segment sizes; 100ms and 1000ms BFD intervals; 40 Mbps, 45 Mbps, 49 Mbps traffic loads with the capacity of 50Mbps. Our results showed that video streaming with a large segment size is more stable than the small segment size for the congestion case. On the other hand, since the BFD interval with 100ms is more sensitive to the traffic load, it detects congestion earlier than the 1000ms interval so that the output of QoE parameters is better than the latter.

In the future, we plan to include reliability of the links into the fault tolerance problem. In this way, we believe that a more robust fault tolerant systems can be designed based on the probability of the link failures. Moreover, we plan to consider the number of DASH clients, the quality switch algorithm used in DASH and the percentage of rerouted flows to investigate their effect on the QoE parameters in case of congestion.

REFERENCES

1. Gozdecki, J., A. Jajszczyk and R. Stankiewicz, "Quality of service terminology in IP networks", *IEEE Communications Magazine*, Vol. 41, No. 3, pp. 153–159, 2003.

2. Cisco Visual Networking Index, *The Zettabyte Era: Trends and Analysis*, 2017, `https://bit.ly/2uBPyaa`, accessed at: June 20, 2018.

3. Cisco Visual Networking Index, "Forecast and methodology 2016-2021", *Cisco White Paper*, 2017.

4. Stockhammer, T., "Dynamic adaptive streaming over HTTP–: standards and design principles", *Proceedings of the second annual ACM conference on Multimedia systems*, pp. 133–144, ACM, 2011.

5. Seufert, M., S. Egger, M. Slanina, T. Zinner, T. Hossfeld and P. Tran-Gia, "A survey on quality of experience of HTTP adaptive streaming", *IEEE Communications Surveys & Tutorials*, Vol. 17, No. 1, pp. 469–492, 2015.

6. Fonseca, P. and E. Mota, "A survey on fault management in software-defined networks", *IEEE Communications Surveys & Tutorials*, 2017.

7. Yu, Y., X. Li, X. Leng, L. Song, K. Bu, Y. Chen, J. Yang, L. Zhang, K. Cheng and X. Xiao, "Fault Management in Software-Defined Networking: A Survey", *IEEE Communications Surveys & Tutorials*, 2018.

8. Sharma, S., D. Staessens, D. Colle, M. Pickavet and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements", *Computer Communications*, Vol. 36, No. 6, pp. 656–665, 2013.

9. van Adrichem, N. L., B. J. van Asten and F. A. Kuipers, "Fast Recovery in Software-Defined Networks", *Third European Workshop on Software Defined Net-*

works, pp. 61–66, IEEE, sep 2014.

10. De Oliveira, R. L. S., A. A. Shinoda, C. M. Schweitzer and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks", *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pp. 1–6, IEEE, 2014.

11. Varis, N., "Anatomy of a Linux bridge", *Proceedings of Seminar on Network Protocols in Operating Systems*, p. 58, 2012.

12. McKeown, N., T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: enabling innovation in campus networks", *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 2, pp. 69–74, 2008.

13. Nunes, B. A. A., M. Mendonca, X.-N. Nguyen, K. Obraczka and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks", *IEEE Communications Surveys & Tutorials*, Vol. 16, No. 3, pp. 1617–1634, 2014.

14. Xia, W., Y. Wen, C. H. Foh, D. Niyato and H. Xie, "A survey on software-defined networking", *IEEE Communications Surveys & Tutorials*, Vol. 17, No. 1, pp. 27–51, 2015.

15. Kreutz, D., F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-defined networking: A comprehensive survey", *Proceedings of the IEEE*, Vol. 103, No. 1, pp. 14–76, 2015.

16. Oyman, O. and S. Singh, "Quality of experience for HTTP adaptive streaming services", *IEEE Communications Magazine*, Vol. 50, No. 4, 2012.

17. Wang, Z., A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity", *IEEE Transactions on Image*

Processing, Vol. 13, No. 4, pp. 600–612, 2004.

18. Georgopoulos, P., Y. Elkhatib, M. Broadbent, M. Mu and N. Race, "Towards network-wide QoE fairness using openflow-assisted adaptive video streaming", *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, pp. 15–20, ACM, 2013.

19. Sharma, S., D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Enabling fast failure recovery in OpenFlow networks", *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*, pp. 164–171, IEEE, 2011.

20. Kim, H., M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner and N. Feamster, "Coronet: Fault tolerance for software defined networks", *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pp. 1–2, IEEE, 2012.

21. Nguyen, K., Q. T. Minh and S. Yamada, "A software-defined networking approach for disaster-resilient WANs", *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pp. 1–5, IEEE, 2013.

22. Li, J., J. Hyun, J.-H. Yoo, S. Baik and J. W.-K. Hong, "Scalable failover method for data center networks using OpenFlow", *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–6, IEEE, 2014.

23. Gude, N., T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown and S. Shenker, "NOX: towards an operating system for networks", *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 3, pp. 105–110, 2008.

24. Yuan, B., H. Jin, D. Zou, L. T. Yang and S. Yu, "A Practical Byzantine-Based Approach for Faulty Switch Tolerance in Software-Defined Networks", *IEEE Transactions on Network and Service Management*, Vol. 15, No. 2, pp. 825–839, 2018.

25. Song, S., H. Park, B.-Y. Choi, T. Choi and H. Zhu, "Control path management

framework for enhancing software-defined network (SDN) reliability", *IEEE Transactions on Network and Service Management*, Vol. 14, No. 2, pp. 302–316, 2017.

26. Desai, M. and T. Nandagopal, "Coping with link failures in centralized control plane architectures", *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*, pp. 1–10, IEEE, 2010.

27. Kempf, J., E. Bellagamba, A. Kern, D. Jocha, A. Takács and P. Sköldström, "Scalable fault management for OpenFlow", *Communications (ICC), 2012 IEEE international conference on*, pp. 6606–6610, IEEE, 2012.

28. Ramos, R. M., M. Martinello and C. E. Rothenberg, "Slickflow: Resilient source routing in data center networks unlocked by openflow", *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pp. 606–613, IEEE, 2013.

29. Ramos, R. M., M. Martinello and C. E. Rothenberg, "Data center fault-tolerant routing and forwarding: An approach based on encoded paths", *Dependable Computing (LADC), 2013 Sixth Latin-American Symposium on*, pp. 104–113, IEEE, 2013.

30. Reitblatt, M., M. Canini, A. Guha and N. Foster, "Fattire: Declarative fault tolerance for software-defined networks", *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 109–114, ACM, 2013.

31. Petroulakis, N. E., G. Spanoudakis and I. G. Askoxylakis, "Fault Tolerance Using an SDN Pattern Framework", *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2017.

32. Cascone, C., D. Sanvito, L. Pollini, A. Capone and B. Sansò, "Fast failure detection and recovery in SDN with stateful data plane", *International Journal of Network Management*, Vol. 27, No. 2, 2017.

33. Sharma, S., D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Fast failure

recovery for in-band OpenFlow networks", *Design of reliable communication networks (drcn), 2013 9th international conference on the*, pp. 52–59, IEEE, 2013.

34. Borokhovich, M., L. Schiff and S. Schmid, "Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms", *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 121–126, ACM, 2014.

35. Van Adrichem, N. L., B. J. Van Asten and F. A. Kuipers, "Fast recovery in software-defined networks", *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pp. 61–66, IEEE, 2014.

36. Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, Jun. 2010.

37. Pfeiffenberger, T., J. L. Du, P. B. Arruda and A. Anzaloni, "Reliable and flexible communications for power systems: Fault-tolerant multicast with sdn/openflow", *New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on*, pp. 1–6, IEEE, 2015.

38. Thorat, P., S. Raza, D. S. Kim and H. Choo, "Rapid recovery from link failures in software-defined networks", *Journal of Communications and Networks*, Vol. 19, No. 6, pp. 648–665, 2017.

39. Zabrovskiy, A., E. Kuzmin, E. Petrov and M. Fomichev, "Emulation of dynamic adaptive streaming over http with mininet", *Proceedings of the 18th Conference of Open Innovations Association FRUCT*, pp. 391–396, FRUCT Oy, 2016.

40. Mkwawa, I.-H., A. A. Barakabitze and L. Sun, "Video quality management over the software defined networking", *Multimedia (ISM), 2016 IEEE International Symposium on*, pp. 559–564, IEEE, 2016.

41. Bentaleb, A., A. C. Begen and R. Zimmermann, "SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking", *Proceedings of the*

2016 ACM on Multimedia Conference, pp. 1296–1305, ACM, 2016.

42. Bentaleb, A., A. C. Begen, R. Zimmermann and S. Harous, "SDNHAS: An SDN-Enabled architecture to optimize qoe in http adaptive streaming", *IEEE Transactions on Multimedia*, Vol. 19, No. 10, pp. 2136–2151, 2017.

43. Bagci, K. T., K. E. Sahin and A. M. Tekalp, "Compete or Collaborate: Architectures for Collaborative DASH Video Over Future Networks", *IEEE Transactions on Multimedia*, Vol. 19, No. 10, pp. 2152–2165, 2017.

44. Lu, Y. and S. Zhu, "SDN-based TCP congestion control in data center networks", *Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance*, pp. 1–7, IEEE, 2015.

45. Kim, S., J. Son, A. Talukder and C. S. Hong, "Congestion prevention mechanism based on Q-leaning for efficient routing in SDN", *Information Networking (ICOIN), 2016 International Conference on*, pp. 124–128, IEEE, 2016.

46. Cheng, Z., X. Zhang, Y. Li, S. Yu, R. Lin and L. He, "Congestion-aware local reroute for fast failure recovery in software-defined networks", *IEEE/OSA Journal of Optical Communications and Networking*, Vol. 9, No. 11, pp. 934–944, 2017.

47. Nasimi, M., M. A. Habibi, B. Han and H. D. Schotten, "Edge-Assisted Congestion Control Mechanism for 5G Network Using Software-Defined Networking", *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, pp. 1–5, IEEE, 2018.

48. Project Floodlight, *Floodlight Controller*, `http://www.projectfloodlight.org/floodlight/`, accessed at December 29, 2018.

49. DASH Industry Forum, *dash.js*, `http://cdn.dashjs.org/latest/jsdoc/index.html`, accessed at December 9, 2018.

50. iPerf Traffic Generator, *iPerf*, `https://iperf.fr`, accessed at December 9, 2018.

51. Wireshark Network Protocol Analyzer, *Wireshark*, `https://www.wireshark.org`, accessed at December 9, 2018.

52. Open Virtual Switch, *Open vSwitch*, `http://www.openvswitch.org`, accessed at December 16, 2018.