Fault Tolerance in SDN Data Plane Considering Network and Application Based Metrics

Baris Yamansavascilar[a,*], Ahmet Cihat Baktir[a], Atay Ozgovde[b], Cem Ersoy[a]

[a]*Department of Computer Engineering, Bogazici University, Istanbul, Turkey*
[b]*Department of Computer Engineering, Galatasaray University, Istanbul, Turkey*

Abstract

Failures in networks result in service disruptions which may cause deteriorated Quality of Service (QoS) for the end-users. Since SDN is becoming the mainstream paradigm for networks, implementation of a robust fault tolerance scheme for SDN-based networks is crucial. Existing SDN data plane fault tolerance approaches can be classified as reactive and proactive which may or may not rely on the controller, respectively. However, none of them qualifies as a complete solution, providing only partial remedies. In this work, we propose Dynamic Protection with Quality of Alternative Paths (DPQoAP) that considers not only the existing faults within the network but also the quality of alternative paths. As a result, we can sustain the QoS throughout the network after the recovery. We also investigate how application based parameters are affected by link 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. On the other hand, even though DASH is proposed as a solution to improve the QoE affected by the dynamic conditions of the networks, it remains insufficient to handle the congested links that show the symptoms of a link failure. 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. The performance of the proposed solutions is evaluated through various experiments considering the QoS and QoE parameters. It is observed that DPQoAP enhances the efficiency of the networking operations and adaptability of the applications.

Keywords:
SDN, Fault Tolerance, Reliability, Dynamic Adaptive Streaming over HTTP (DASH), Quality of Experience (QoE), Quality of Service (QoS)

1. Introduction

Alleviating failures is crucial for service providers in order to meet the Quality of Service (QoS) expected by their clients (Gozdecki et al., 2003). Strict Service Level Agreement (SLA) requirements when combined with the risk of reputation loss are evident that fault handling should be carefully accomplished. Ideally, network failures should be handled seamlessly, transparent to the end-user, not affecting their Quality of Experience (QoE). Practically, however, remedies try to 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 operations. This impact can be broadly categorized as the impact on QoS behavior and the effect on QoE behavior of the network. Here, the QoS parameters represent the application-independent network characteristics whereas QoE parameters represent the specific impact that 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.

In the context of fault tolerance, Software-Defined Networking (SDN) provides important opportunities 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 gradually affect other applications. In this study, we focus on the fault tolerance within the SDN data plane.

Restoration and protection are two essential approaches for the failure recovery in the data plane. Both of these approaches have their respective advantages and disadvantages in terms of the recovery time and recent network view (da Rocha Fonseca and Mota, 2017; Yu et al., 2018; Saraswat et al., 2019). In the restoration, the new routing rules for the affected flows are computed considering the recent network view. During this process, the controller itself is the responsible entity as shown in Figure 1. 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. On the other hand, in the protection, the alternative rules for active flows are predefined

*Corresponding author.
Email addresses: baris.yamansavascilar@boun.edu.tr (Baris Yamansavascilar), cihat.baktir@boun.edu.tr (Ahmet Cihat Baktir), aozgovde@gsu.edu.tr (Atay Ozgovde), ersoy@boun.edu.tr (Cem Ersoy)

for possible failures in the future. 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.

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 substituting 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. With this vision in mind, we propose Dynamic Protection with Quality of Alternative Path (DPQoAP) which combines both the restoration and the protection methods to achieve fast recovery while selecting among available high-quality paths. To achieve this goal, DPQoAP takes two important requirements into account: (1) the recent state of the network considering restoration, and (2) backup path information for the active flows regarding protection. To carry out the first requirement, DPQoAP periodically checks the quality of alternative paths in the network. For the second requirement, it uses Fast Failover groups to hold backup path information in corresponding switches.

To evaluate the performance of DPQoAP, we focused on the video streaming use case. Currently, video streaming is the most influential traffic type of the Internet since it occupies 82% of the overall data traffic volume (Cisco Visual Networking, 2020). 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. Dynamic Adaptive Streaming over HTTP (DASH) is the widely accepted technology recently for video streaming. DASH, can receive video segments in independent connections, which in turn allows for versatile and dynamic streaming operation.

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. Thus, we combine the capabilities of DASH and our novel approach that perceives the congestion as the fundamental element for the link failure. Our experimental results show that QoE parameters including video quality, bitrate latency, and the number of quality switches are improved dramatically when we apply our method to the congestion case.

Another focus of our work is related to the inadequacy of an experimentation method that is typically being used in the literature. Mininet, which owes its reputation to the wide variety of its capabilities, is the emulation environment for evaluating SDN-based proposals (De Oliveira et al., 2014). However, in the context of generating link failures, we argue that capabilities provided by Mininet are insufficient to emulate a fault scenario in a realistic manner 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 almost zero delay which can cause incorrect recovery times to be reported. As a remedy, in our work, we incorporate L2 Linux bridges (Varis, 2012) 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.

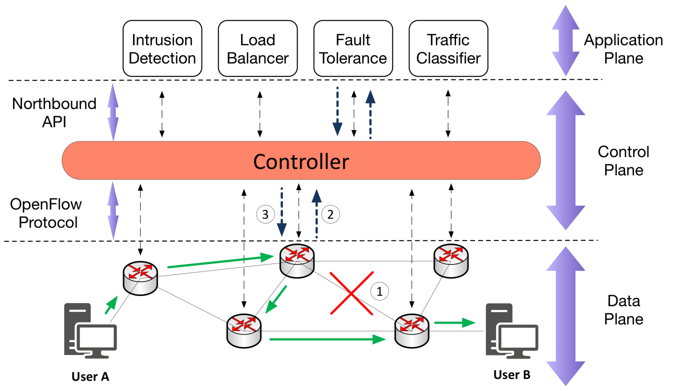


Figure 1: Restoration in SDN.

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

- We propose DPQoAP which is a novel dynamic protection method considering the quality of alternative paths. In this case, in addition to the recovery time, we also consider the alternative paths for flows affected by the failures.
- We investigate how video quality and thus QoE would fluctuate when a failure occurs. 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 L2, 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 Bidirectional Forwarding Detection (BFD) protocol rather than the application layer metrics that is generally used by video applications to adapt themselves for new conditions.

Detecting congestion at the network level through SDN allows us to implement a global solution as opposed to each client adapting 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 environments. 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 is beneficial for the congestion case.

The rest of this study is organized as follows. In Section 2, we elaborate the related works including DASH, fault tolerance, and congestion studies that use SDN as their networking paradigm. Section 3 provides the background information about OpenFlow groups, BFD, and dynamic adaptive video streaming with QoE. Section 4 provides details of our fault tolerant design. Afterwards, in Section 5 and 6, we present the performance evaluation of our system along with the experiments. Finally, we conclude this study in Section 7. We give the primary notations used throughout the paper in Table 1.

Table 1: List of abbreviations

Notation	Description
ACK	Acknowledgment
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
DPQoAP	Dynamic Protection with Quality of Alternative Paths
FF	Fast Failover
HTTP	Hypertext Transfer Protocol
L2	Link Layer 2
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
VLAN	Virtual Local Area Network
VoIP	Voice over IP
WAN	Wide Area Network

2. Related Works

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 study focuses on the data plane domain as it is the level where the actual traffic is handled.

2.1. 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 (Sharma et al., 2011; Kim et al., 2012; Nguyen et al., 2013; Li et al., 2014) 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. (Kim et al., 2012) used VLANs to compute routing paths. In (Sharma et al., 2011), Sharma et al. compared their fast failover system with the Learning Switch, Learning PySwitch, and Routing Mode of the NOX controller (Gude et al., 2008). Nguyen et al. (Nguyen et al., 2013) 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. (Li et al., 2014) on the other hand developed a restoration approach by using a local optimal failover method in order to reduce the path calculation time. In (Zhang et al., 2011), Zhang et al. investigated network resilience of OpenFlow based centralized networks by considering connectivity failures and controller placement problem. They first formulated the controller placement metrics and failures including link, switch and switch-to-controller. Afterwards they performed resilience analysis of the centralized network architecture. Their experimental results showed that the location of the controller affects resilience of the network significantly. Lee et al. (Lee et al., 2019) focused on real-time flows considering their own fault tolerance constraints using adaptive path restoration. They implemented a multi-constrained path finding algorithm which can reroute the flows based on their time budget. In (Tajiki et al., 2019), authors considered failure recovery with service function chaining. Based on the service requirements of each service, they used restoration as a fault-aware routing.

Some studies also considered reliability in this manner. In (Yuan et al., 2018) Yuan et al. designed a system based on the

Byzantine model to tolerate faulty switches in order to enhance reliability. Moreover, Song et al. (Song et al., 2017) 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. Besides, Bhatia et al. (Bhatia et al., 2019) considered reliability in an SDN-enabled VANET environment. They used a network coding based approach for reliable data dissemination.

2.2. 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 (Desai and Nandagopal, 2010), authors built a system that permits switches to send faulty link information to only the relevant switches in order to prevent traffic flooding 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. (Kempf et al., 2012) 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 ms using this function. In (Ramos et al., 2013b), Ramos et al. extended their previous study (Ramos et al., 2013a) 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.

Beheshti et al. (Beheshti and Zhang, 2012) investigated the fault tolerance between switch and controller connection by considering controller placement and control-traffic metrics. They evaluated the impact of the routing and placement algorithms on several topologies. They formulated the resiliency problem and propose a protection method for the connectivity between the controller and the switches. In (Zhu and Lan, 2015), authors proposed a protection mechanism for node failures in SDN by applying a backup flow-table architecture using ant colony algorithm. For each main path, they deposited an alternative (backup) path in case there is a failure. Gyllstrom et al. (Gyllstrom et al., 2014) addressed reliable multicasting of critical Smart Grid data. To this end, they designed a link failure detection algorithm namely PCOUNT, and formulated an optimization problem for computing multicast trees. Moreover, they tested proactive and reactive schemes. On the other hand, controller placement and switch migration in a multi-controller SDN environment are important to provide fault tolerance considering protection. To this end, (Al-Tam and Correia, 2019) and (Correia and Faroq, 2019) can be evaluated as link-protection preplanning.

On the other hand, Reitblatt et al. (Reitblatt et al., 2013) 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. (Petroulakis et al., 2017) proposed a pattern framework for fault tolerance using a rule-based language. Moreover, Cascone et al. (Cascone et al., 2017) 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. (Sharma et al., 2013b) considered carrier-grade networks in which fault recovery should be completed in 50 ms, 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 (Sharma et al., 2013a), 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 (Borokhovich et al., 2014), 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. (Van Adrichem et al., 2014) used Bidirectional Forwarding Detection (BFD) protocol (Katz and Ward, 2010) per link as well as (Sharma et al., 2013b) to detect failures and then compared the performance of different BFD detection intervals in terms of milliseconds. Pfeiffenberger et al. (Pfeiffenberger et al., 2015) on the other hand, focused on the robust multicasting in SDN by considering fault tolerance. In (Thorat et al., 2017), the authors used VLAN tags in their design in order to reduce alternative path rules.

The features of the highlighted studies that focuses on the data plane fault tolerance problem are given in Table 2.

2.3. DASH and QoE in SDN

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. (Zabrovskiy et al., 2016) 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. (Mkwawa et al., 2016) 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 (Bentaleb et al., 2016), Bentaleb et

Table 2: The features of highlighted studies that focused fault tolerance in SDN

Study	Goal	Method	Main Contribution	Testing	Controller
(Desai and Nandagopal, 2010)	Finding solution for link failures in data plane without involving the controller	Informing relevant switches	Devising a way such that only the necessary switches are informed of the link failure	Custom Emulation	Custom
(Sharma et al., 2011)	Fast Recovery	Restoration	Comparing different restoration options in OpenFlow networks	Custom Emulation	POX
(Kim et al., 2012)	Scalable Fast Recovery	VLAN usage with restoration	A fault-tolerant SDN architecture scaling to large network sizes	Mininet	NOX
(Kempf et al., 2012)	Failure detection without involving the controller	Extending OpenFlow 1.1 protocol	Extending OpenFlow protocol to support monitoring function in data plane by using switches	Custom	NOX
(Ramos et al., 2013b)	Proactive Failure Recovery	Alternative routes carried in packet headers	Alternative routes carried in packet headers by using their OpenFlow prototype	Mininet	NOX
(Reitblatt et al., 2013)	Fault-tolerant SDN using regular expressions	Regular expressions and OpenFlow FF Groups	A new language for writing fault tolerant programs	Mininet	Custom
(Sharma et al., 2013a)	Failure recovery for in-band OpenFlow network	Restoration and Protection	Evaluation of failure recovery considering both control and data traffic for in-band network	Mininet	NOX
(Li et al., 2014)	Recovery of link failures	Restoration with local optimal failover	Scalable failover method in data center networks	Mininet	Floodlight
(Petroulakis et al., 2017)	Rule-based language in order to provide fault tolerance in SDN networks	SDN Pattern Framework	Fault tolerance patterns that can provide restoration and protection failures	Mininet	ODL
(Cascone et al., 2017)	Fast recovery	Protection using finite state machines in switches	Using stateful approach (finite state machines) in data plane for fast detection and recovery	Mininet	Ryu
(Thorat et al., 2017)	Fast recovery with aggregated flows	OpenFlow FF groups with VLAN	Efficient memory usage with minimum controller intervation	Mininet	NOX
(Yuan et al., 2018)	Automatically tolerate faulty switches	Byzantine fault tolerance model	A system that guarantees the correctness of flow statistics information even when faulty switches exist	Mininet	RYU
(Lee et al., 2019)	Fault-resilient real-time networking system	Adaptive path restoration	A system that supports real-time flows with their own fault tolerance constraints	Physical Network Testbed	Odroid-XU4
(Tajiki et al., 2019)	Failure recovery with service function chaining	Restoration regarding service requirements	Fault-aware routing architecture for service function chaining problems with energy consideration.	Custom Emulation	Custom

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 (Bentaleb et al., 2017), considering scalability issues of clients, communication overhead, and client heterogeneity. Likewise, Bagci et al. (Bagci et al., 2017) studied QoE fairness among clients. They used the network slicing concept and manipulated TCP windows to prevent QoE fluctuations.

2.4. Congestion in SDN

SDN is also used for the congestion case by several studies. In (Lu and Zhu, 2015), 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. (Kim et al., 2016) 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. (Cheng et al., 2017) 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 (Nasimi et al., 2018), 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. In (Bhatia et al., 2020), the authors performed a traffic congestion analysis considering SDN-based real-time urban traffic analysis in VANET environment. Since their primary goal is to enhance traffic flow prediction in a VANET environment, they also focused to find congestive sensitive spots. To perform traffic congestion prediction, they used a LSTM model.

2.5. Differences Between Existing Works and Ours

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 such as those listed in Table 2 considered only QoS parameters including recovery time, delay, and the number of affected flows rather than the 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. Furthermore, we investigate the Linux bridge effect on fault management considering BFD-based solutions for both QoS and QoE. Thus, our study is distinctly separated from the existing works.

3. Background

Since the background information is essential to comprehend our proposed fault tolerant scheme, we provide the fundamental explanations about the OpenFlow group concept, BFD, and DASH in this section, respectively.

3.1. 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 type, counters, and a list of action buckets as shown in Figure 2a. 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:

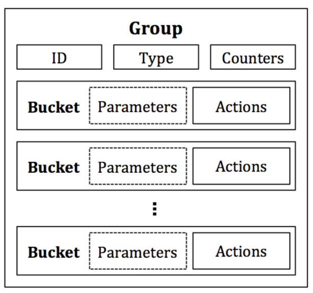
- **All Group:** This group is used for multicasting or broadcasting. A packet in this group is copied for each bucket in the bucket list and handled independently.

- **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.

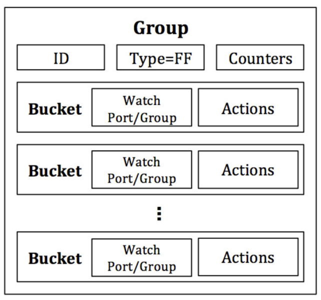
- **Indirect Group:** There is only one bucket in this group type. The goal is to cover commonly used actions for the same next hop when forwarding and thus reduce the switch memory utilization.

- **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 2b.

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 3. 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.



(a) Standard OpenFlow group.



(b) OpenFlow fast failover group.

Figure 2: OpenFlow Groups.

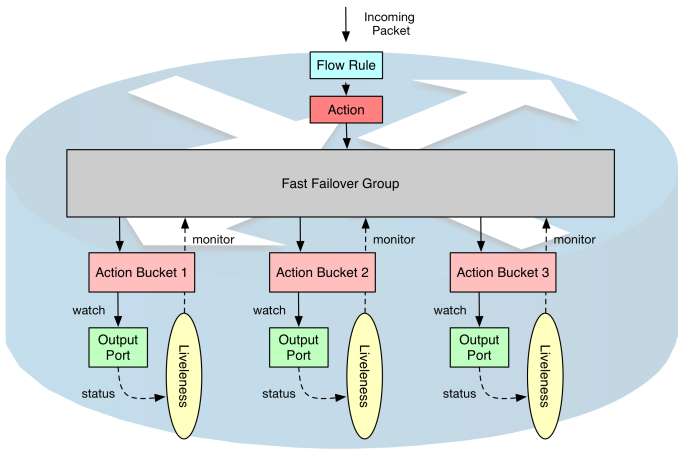


Figure 3: The working mechanism of a Fast Failover group in OpenFlow switch.

3.2. *Bidirectional Forwarding Detection (BFD)*

Among the failure detection methods, the BFD protocol is special since it is designed specifically for failure detection and its detection speed outperforms the others (Sharma et al., 2013b; van Adrichem et al., 2014). 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 1. The detection time multiplier is used to prevent false positives that can occur due to packet loss.

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

3.3. *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) behave 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.

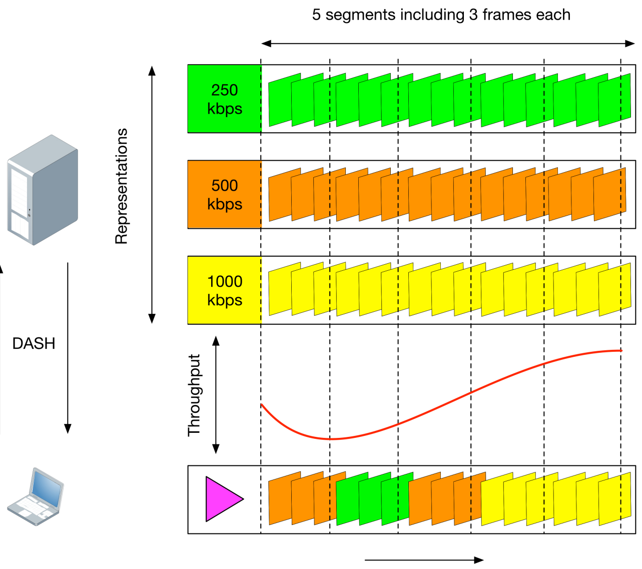


Figure 4: Concept of DASH (Adapted from (Seufert et al., 2015)).

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 (Stockhammer, 2011). 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 4.

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

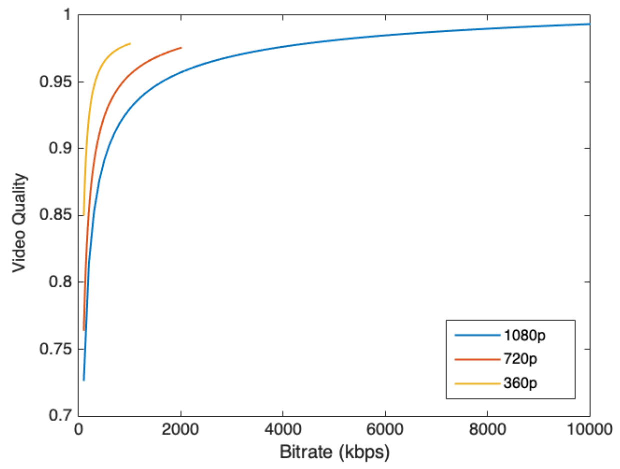


Figure 5: Relation between video quality and bitrate for three different resolutions.

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 (Oyman and Singh, 2012) including HTTP 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) (Wang et al., 2004). 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 5, the mapping of the bitrate to the video quality using SSIM demonstrated that the relationship between bitrate and perceptual quality is not linear (Georgopoulos et al., 2013). Hence, using three resolution types and various bitrates given in Table 3, a generalized function for QoE based on bitrate and resolution is configured by conducting curve fitting (Georgopoulos et al., 2013). The formula is given in Equation 2 and corresponding coefficients are given in Table 4. In the equation, *f(x)* represents the video quality, and variable *x* denotes the bitrate.

$$f(x) = ax^b + c \tag{2}$$

4. Design of the Fault Tolerant Data Plane

In our fault tolerant data plane design, we implement our novel method, DPQoAP, along with other fault tolerance approaches including static protection, and restoration by considering their use cases and to observe the performance of our proposal. Moreover, we apply DASH and BFD-based Congestion Detection modules that also exploit the benefits of our fault tolerance package considering end user applications in SDN. The graphical abstract shown in Figure 6 summarizes our design.

Table 3: 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 4: 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

4.1. DPQoAP 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 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 DPQoAP, 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 controller (Floodlight Controller, 2019) provides an API call to obtain the latency values of the links, this information is used for each path to determine the best alternative path for every QoAP calculation interval, T_{qoap}. A demonstrative example is shown in Figure 7. After the formation of the primary path and two backup paths as depicted in Figure 7a, the secondary path, Path 2, becomes loaded with a recently generated heavy traffic as shown in Figure 7b. This situation is detected by our DPQoAP 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 7c, 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 as given in Algorithm 1. 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 cor-

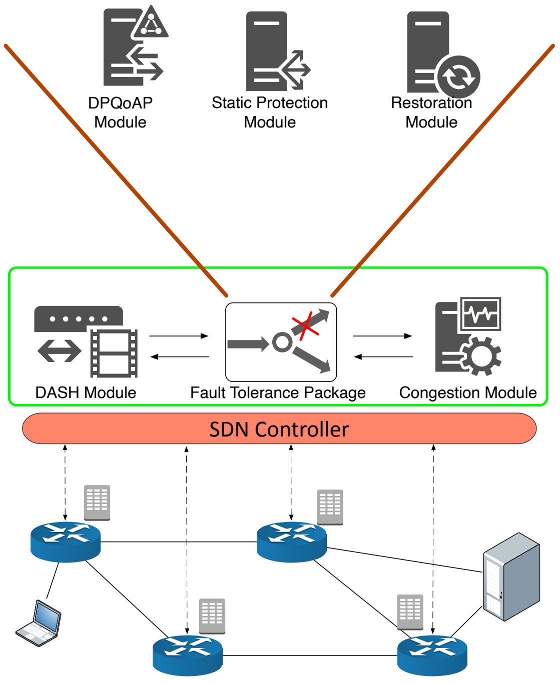


Figure 6: Abstract of our fault tolerant design in SDN.

responding *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.

4.2. DASH Module

To calculate the necessary metrics including the buffer level and the bitrate for the video quality value, we implemented a video client module using DASH.js (dash.js, 2019). 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 and congestion.

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.

4.3. 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 8 shows the main aspects of our design.

ALGORITHM 1: DPQoAP Module

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

4.3.1. 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 8. 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.

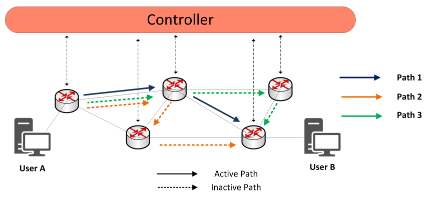
The failure detection time, T_d, is based on the BFD message transmit interval, T_i, that can be manually configured by the network administrator considering the given services. For example, if real-time applications such as VoIP are widely used in the network, the interval must be very low considering the 50 ms recovery time (Sharma et al., 2013b). 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 1. M value is used to prevent false positives.

4.3.2. Rerouting Flows

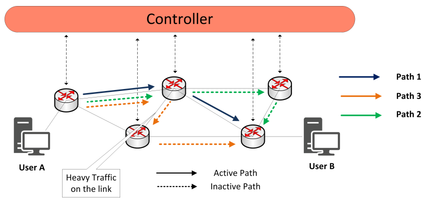
After detecting the congestion using BFD, the problem on the link is reported to the controller via the OpenFlow protocol. Subsequently, the controller informs the modified fault tolerance application about the problem. In the application, flows passing through that link are first extracted from the flow pool in which all active flows are held. Afterwards, the predefined percentage of flows, which is 50% in this study, are selected for rerouting and the new rules are created for them. As a result, the congested link is relieved.

4.4. Restoration Module

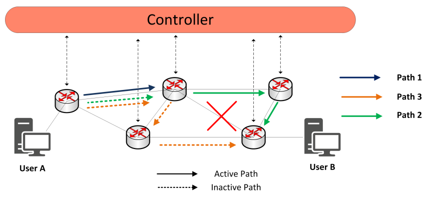
In the restoration, the essential idea is the involvement of the controller. Because of the dynamic nature of the network



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



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



(c) Evading the failure in the network considering the QoAPs.

Figure 7: The concept of the dynamic protection module.

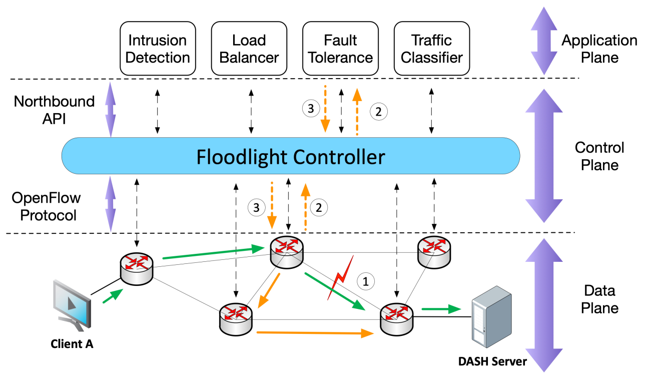
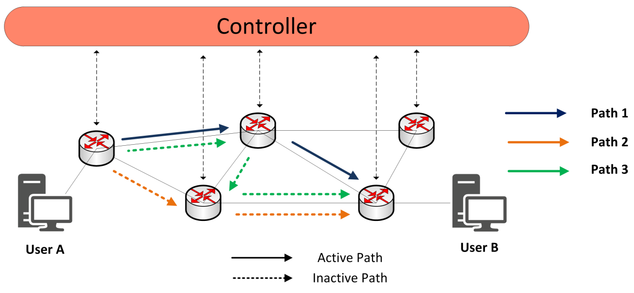
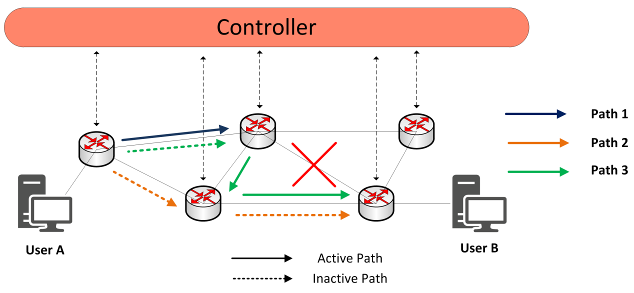


Figure 8: The design of the BFD-based Congestion Detection System.



(a) Primary and backup paths between communicators.



(b) Static protection using OpenFlow groups.

Figure 9: Static Protection in SDN.

environment, the restoration approach is important to handle the failures in the network. We implemented the restoration module as demonstrated in Algorithm 2 by applying the steps shown in Figure 1. 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.

ALGORITHM 2: The Algorithm of Restoration Module

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

4.5. 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 9. 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 9b, 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.

ALGORITHM 3: Static Protection Module

```
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 $path \in allPaths$ do
        InstallProactiveRules($path$, $groupID$)
    end
end
Function InstallProactiveRules(path, groupID)
    $index$ = $path$.length();
    for $switch \in path$ do
        $outPort$ = $path$.get($index$);
        $key$ = getKey(switch + groupID);
        $buckets$ = getBucketList($key$);
        if $buckets == null$ then
            $buckets$ = createBucketList();
        end
        if $buckets.contain(outPort) = false$ then
            $buckets$.add($outPort$);
        end
        $switch$.update($buckets$);
    end
end
```

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, Algorithm 3 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.

5. Performance Evaluation of Network-based Metrics

In this section, we evaluate the network-based metrics for QoS in fault tolerance. We conducted several experiments using iPerf (iPerf, 2019) and Wireshark (Wireshark, 2019) tools on several scenarios. To this end, we first evaluated the performance of our DPQoAP module comparing it with the traditional protection approach. Afterwards, 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 packet loss.

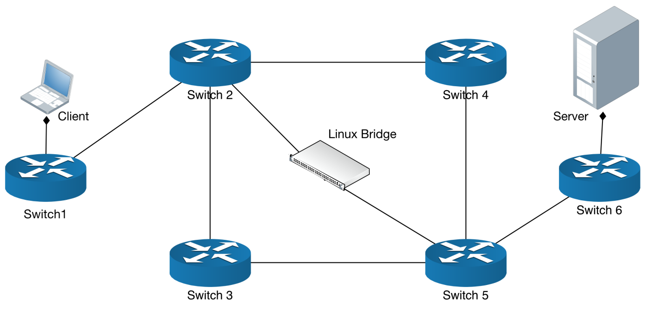


Figure 10: The topology using a Linux bridge.

We repeated experiments 10 times for the variance control. We used Mininet for the SDN emulation, and Open vSwitch (Open vSwitch, 2019) for virtual switches created in Mininet.

In the experiments, based on the scenario, we trigger the failure in two ways:

1. *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.

2. *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.

Considering the Linux bridge command, we used the topology depicted in Figure 10. We also employed the same topology without deploying Linux bridge for the usage of the Mininet command.

5.1. Evaluation of the DPQoAP

For a proper assessment of DPQoAP, we first compare our module with the static protection module. In the test scenario, we create 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 the dynamic protection.

In our experiments each of which lasts 50 seconds, we used the topology shown in Figure 10 without deploying Linux

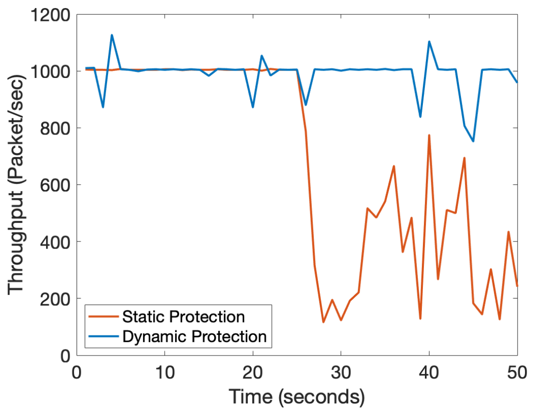


Figure 11: A throughput difference between the static and dynamic protection approaches considering QoAPs.

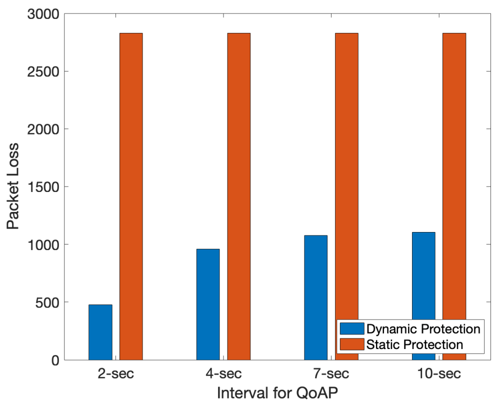


Figure 12: Comparison of the dynamic and static protection approaches based on the packet loss for each QoAP interval.

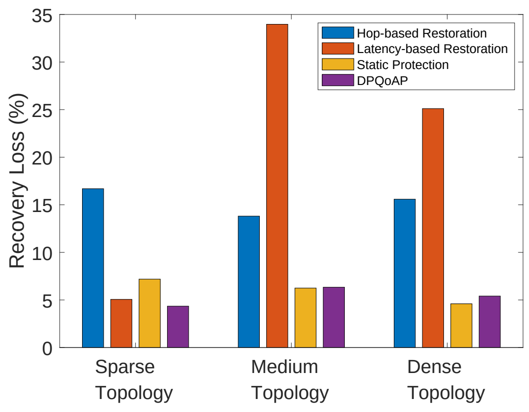


Figure 13: Percentage of loss during 2 seconds of window when the failure occurs.

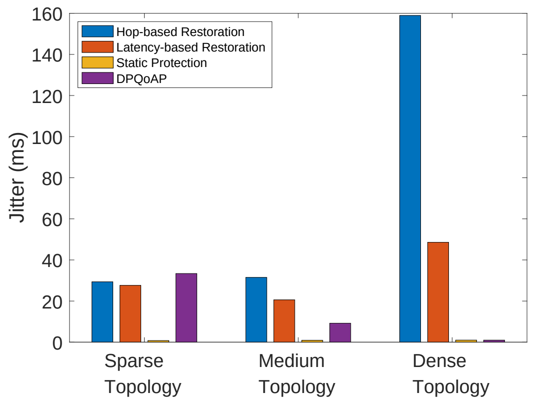


Figure 14: Average jitter of the flow during 20 seconds of window after the failure.

bridge. 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 Switches 2 and 5.

The result shown in Figure 11 depicts that the throughput of the transmission significantly reduces when the QoAP is not considered. Moreover, using our DPQoAP module, 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 12 presents that our dynamic protection approach outperforms 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 DPQoAP decreases, the packet loss reduces since the application uses more recent information of the network. However, lower interval causes higher cost for the system since the statistics of 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.

To generalize the performance of DPQoAP, we conducted three experiments considering three different topologies each of which has different complexities in terms of possible paths after the failure. The sparse topology is the same as Figure 10 that has two possible paths after the failure between Switches 2 and 5. Similarly, the medium topology has 14 possible paths after the failure, and the dense topology has 29 possible paths. We repeated the experiments 30 times for each scenario to eliminate the randomization and used the average value of them.

In these experiments, we also included the two versions of the restoration approach based on the hop-count and the latency. We first investigated the recovery loss that shows what fraction of the total traffic is affected during 2 seconds of recovery window. Figure 13 shows that restoration approaches have poor performance for the recovery loss since the controller is involved in solving the problem. Moreover, since the calculations in latency-based restoration are more complex, hop-based restoration provides better results. Considering DPQoAP and the static protection on the other hand, they produce similar

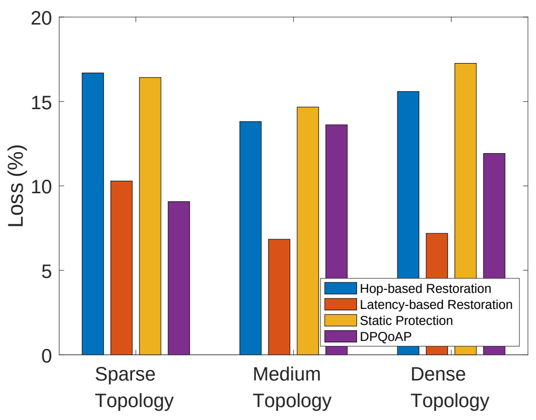


Figure 15: The percentage of the total loss of the flow during 20 seconds of window after the failure.

results for different topologies since the alternative paths have already been calculated before the failure.

Another important measurement for QoS is the jitter since it directly affects the quality of real-time applications like VoIP. To this end, we also compared the performance of the different approaches based on different topologies. Figure 14 shows that when the complexity of the topology increases, the jitter decreases if DPQoAP is used. The main reason of this result is that since there are more possible paths in dense topology, the quality of the selected path in them does not change in general. However, since the possible options are limited in sparser topologies, DPQoAP switches the selected path more frequently so that it causes higher jitter. On the other hand, static protection provides lower jitter for each topology type since the paths are calculated once before the failure, and it does not change the selected path. Besides, the jitter of the restoration approaches increases based on topology complexity due they face more complex calculations for the new path.

The result of the last experiment that considers the percentage of the total loss of the flow during 20 seconds long window after the failure is given in Figure 15. This experiment is also a generalized version of the throughput result given in Figure 11. The results validate that our DPQoAP module is better than the static protection approach since it considers QoAPs. However, since the latency based restoration selects the new path based on the latest condition of the network, it gives the best result in terms of total loss after the failure. This is an expected result considering the working mechanism of the approaches. Since the controller traffic is crucial in SDN, using the restoration approach in a heavy network traffic environment may cause problems in the data plane. To show this effect, we reported the control path traffic considering the different number of flows affected by the failure. The results shown in Figure 16 and Figure 17 present the tradeoff between the restoration and 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 16a, 16b, 16c, and 16d. 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, the increasing number of affected flows may cause significant disruptions.

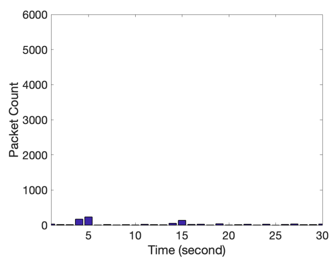
On the other hand, the protection approaches do not affect the control path traffic when the failure occurs as shown in Figures 17a, 17b, 17c, and 17d since the alternative paths have already been configured in the switches in the case of a failure on the link. As a result, a network that includes thousands of flows may not be affected by a link failure in contrast to the restoration approach. However, since the 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 protection approaches perform better than the restoration approach in the case of multiple flows.

5.2. Evaluation of Linux Bridge and BFD Intervals

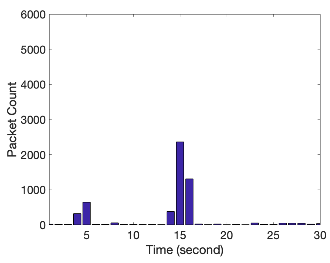
To evaluate the effect of the Linux bridge command in order to observe the broken link scenario in Mininet, we first compared the transmission patterns of the link failures carried out by Mininet command and Linux bridge command, which consists of shutting down a port of Linux bridge, respectively. We applied the restoration and static protection approaches for the evaluation. We used the topology shown in Figure 10 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 18a 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 40 ms. 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 realizes 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 in this case is as shown in Figure 18b. 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.

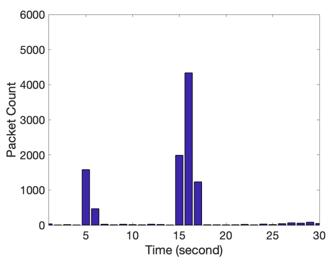
We applied the same approaches above for the evaluation of the static protection module. Since this module solves the problem in the data plane using OpenFlow Fast Failover groups, the recovery time for the link failure created by the Mininet command is smaller than the restoration module with the mean



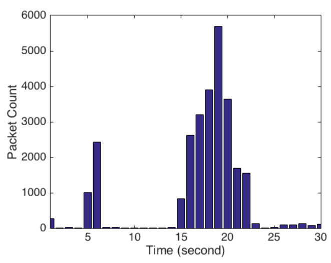
(a) Control path traffic for restoration with 10 flows



(b) Control path traffic for restoration with 20 flows

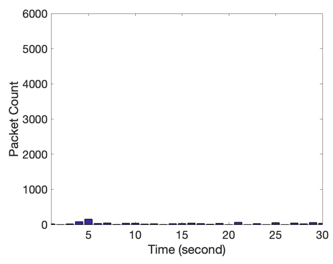


(c) Control path traffic for restoration with 40 flows

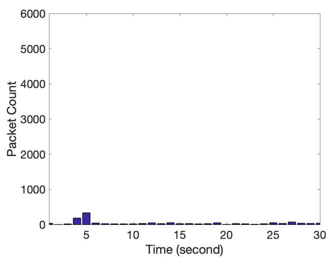


(d) Control path traffic for restoration with 80 flows

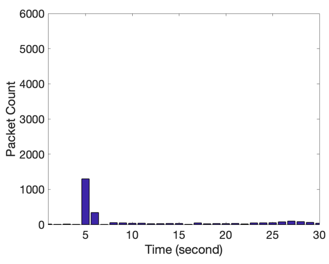
Figure 16: The number of packets of the control path traffic for the restoration approach using 10, 20, 40, and 80 flows during the lifetime of the experiment.



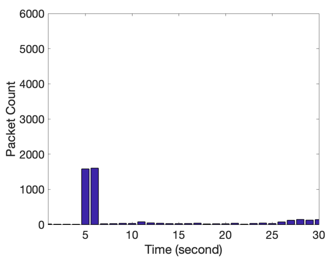
(a) Control path traffic for static protection with 10 flows



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

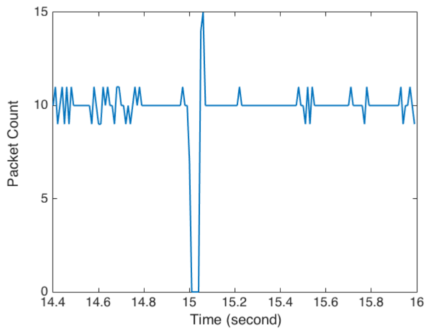


(c) Control path traffic for static protection with 40 flows

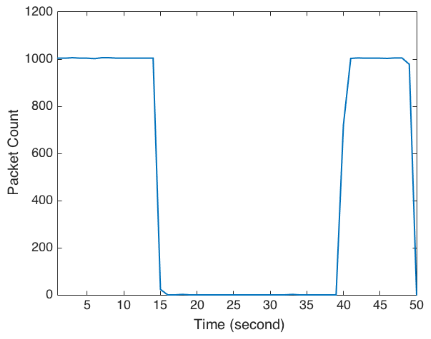


(d) Control path traffic for static protection with 80 flows

Figure 17: The number of packets of the control path traffic for the protection approach using 10, 20, 40, and 80 flows during the lifetime of the experiment.



(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 18: The demonstrative examples of failure recovery for Mininet command and Linux bridge based scenarios.

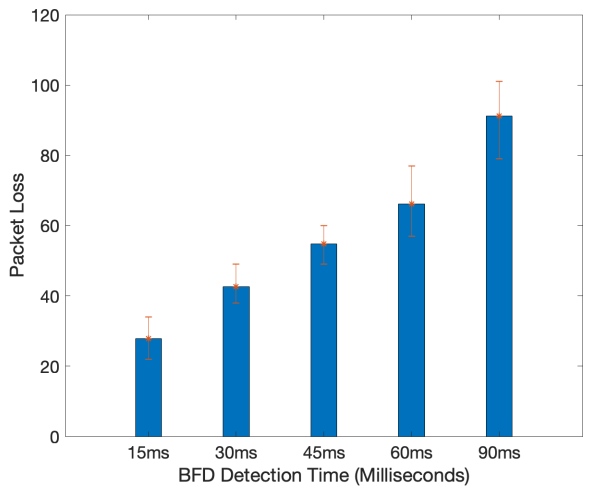


Figure 19: Packet Loss based on the BFD message interval.

value of 28 ms. However, on the other hand, if we shut down the port of the Linux bridge for the failure, the transmission of the packets halt. Since the relevant ports of Switch 2 and 5 are not affected by the failure in this case, OpenFlow FF groups continue to operate due to it checks the liveliness of the ports 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 works based on the value of T_i.

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 15 ms, 30 ms, 45 ms, 60 ms, and 90 ms failure detection times regarding Equation 1 and measured the packet loss. The result shown in Figure 19 presents that when T_i increases, the packet loss also increases since the duration of the failure detection is prolonged.

6. Performance Evaluation of Application-based Metrics

Evaluating the change of the application-based metrics is crucial as well as assessing the network-based parameters since end-users are affected in this case. To this end, we focused on the change of QoE parameters in two ways: (1) when the failure happens and (2) when there is a congestion in the network.

6.1. The Effect of Link Failure on QoE

Our scenarios for this evaluation are similar to the network-based assessment considering the performance of the restoration, static protection and DPQoAP modules in the case of a link failure. In the experiments, we used the topology shown in Figure 10 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 while the buffer level information was provided by DASH API.

6.1.1. 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 20a and 20b. 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 20c and 20d. 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.

6.1.2. 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 halts 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 faulty link, the buffer level and the video quality is not affected by the failure as shown in Figure 20a and 20b.

6.1.3. DPQoAP

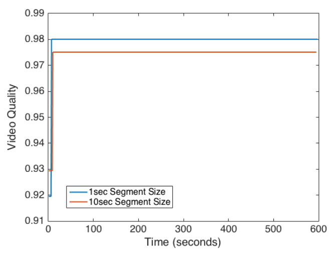
To evaluate the performance of the dynamic protection for the video application, we applied a similar scenario carried out in Section 5.1. We created a heavy traffic on the secondary path at the 100th second before the failure. Afterwards, the link failure was created on the primary path, between Switch 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 21a and 21b 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 DPQoAP 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.

6.2. The Effect of Congestion on QoE

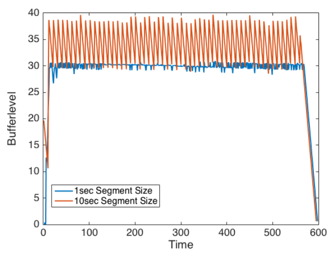
We investigated three factors in our experiments for the congestion case: the impact of the BFD interval, the traffic load, and the video segment size on the QoE parameters. For each experiment, we used Mininet for SDN emulation deploying Open vSwitch for switches since it supports both BFD and OpenFlow 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 22 was used. We limited the capacity of all links as 50 Mbps 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 22 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.

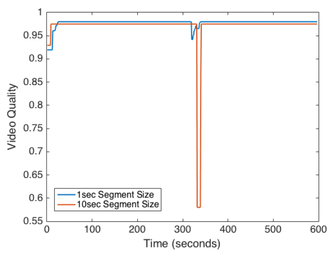
We used 1 second and 10 seconds segments to evaluate the effect of the segment size. To observe the impact of the traffic load, we generated 40 Mbps (80% Load), 45 Mbps (90% Load), and 49 Mbps (98% Load) traffic using only the iPerf clients. Finally, to evaluate the influence of the BFD intervals in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



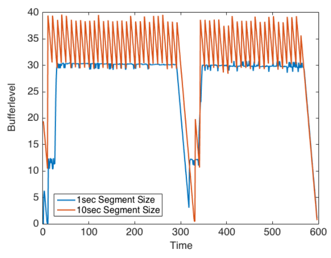
(a) Video quality value when we use Mininet command for the failure.



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

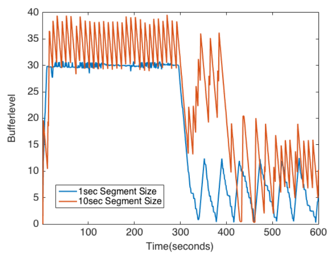


(c) Video quality value is affected when we use Linux bridge for the failure.

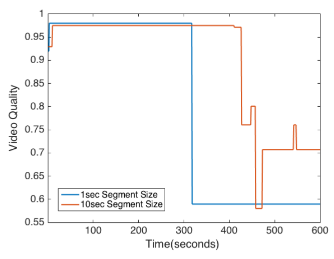


(d) Buffer level is affected when we use Linux bridge for the failure.

Figure 20: The effect of failure recovery on the video quality and buffer level of video streaming using restoration module with Mininet-based and Linux bridge-based failure.



(a) Buffer level without considering QoAP.



(b) Video quality value without considering QoAP.

Figure 21: QoE is significantly affected if the quality of the alternative paths is not considered for fault tolerance.

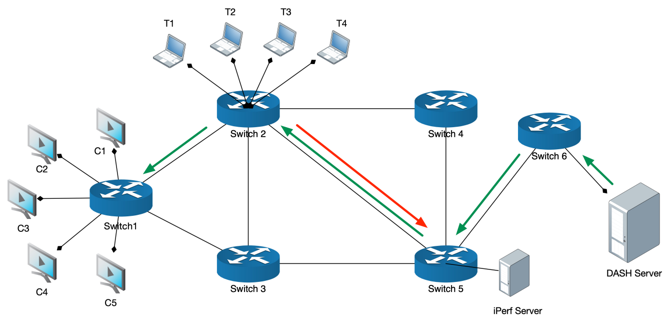


Figure 22: The topology and video traffic route

3000 ms respectively. Moreover, we compared these results with the non-BFD case.

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.

6.3. *Effect of Segment Size*

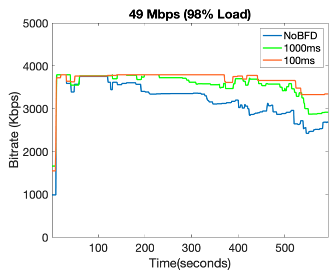
Our experiments showed that streaming with the big segment size is more stable than the small segment size considering the congestion conditions on the link. Experimental results for 49 Mbps (98% Load) traffic on the congested link given in Figure 23 and Figure 24 represent that fluctuations and change of the QoE parameters including the average bitrate, video quality, latency, and number of quality switches between representations are higher in 1 second segment size compared with 10 seconds 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.

6.4. *The Effect of Traffic Load*

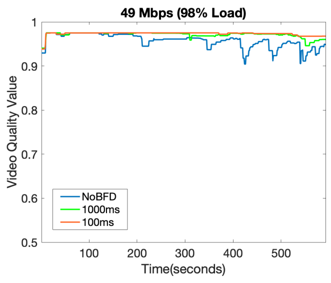
To evaluate the impact of the traffic load, we measured the average video quality of clients based on SSIM and the number of quality switches including all cases in our experiments. Our results demonstrated that when the traffic load increases, the video quality decreases considering both segment sizes with the non-BFD case as shown in Figure 25a and 25b. For each traffic load, the video quality with 10-sec segment size is better than the 1 second 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 second 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 second segment size while it is acceptable for 10-sec segment size.

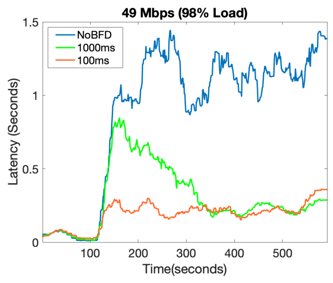
On the other hand, the effect of the traffic load on the number of quality switches is shown in Figure 26a and 26b. The results show that the quality switch count between representations for the 10-sec segment size is the lowest for 80% load and highest for the 90% load when the BFD is not used. This 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 second segment size, the count of switches between representations decrease for higher traffic loads since they cause worse video quality respectively.



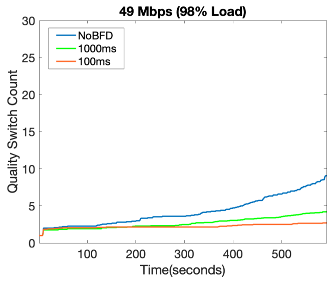
(a) Average bitrate for 10-sec segment size



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

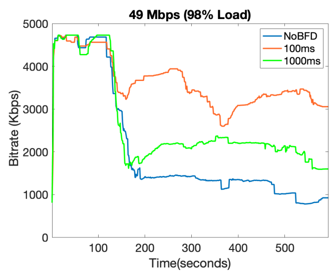


(c) Average latency for 10-sec segment size

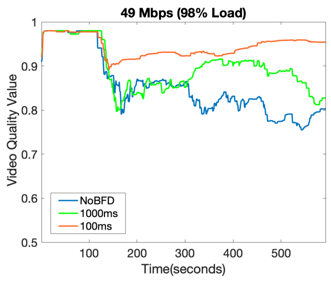


(d) Average number of quality switches for 10-sec segment size

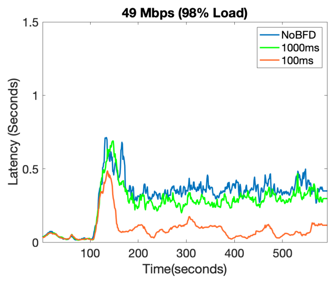
Figure 23: The change of QoE parameters for the congested link with 98% load. The QoE parameters are affected by the congestion after 150th second. If our scheme is used with the BFD mechanism, the QoE parameters are improved for each case.



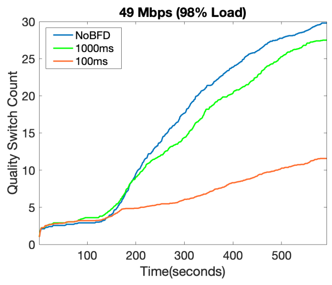
(a) Average bitrate for 1-sec segment size



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

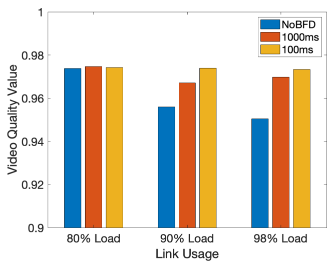


(c) Average latency for 1-sec segment size

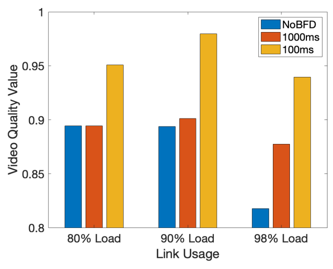


(d) Average number of quality switches for 1-sec segment size

Figure 24: The change of QoE parameters for congested link with 98% load. The QoE parameters are affected by the congestion after 150th second. If our scheme is used with BFD mechanism, the QoE parameters are improved for each case.

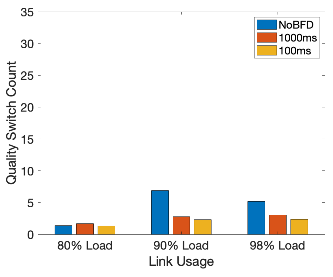


(a) The impact of the traffic load on the average video quality using 10-sec segment size

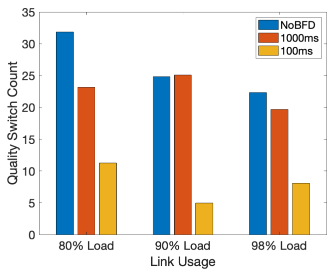


(b) The impact of the traffic load on the average video quality using 1-sec segment size

Figure 25: 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 26: The impact of the traffic load on the quality switch count with respect to different segment sizes.

6.5. *The Effect of BFD Intervals*

Our results showed that the impact of BFD is crucial in the case of congestion. In Figures 23 and 24, the results show that using BFD fixes the poor outputs of each 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 100 ms outperformed T_i with 1000 ms regarding QoE parameters since its sensitivity is more delicate so that it detects the congestion earlier. Apart from the 49 Mbps shown in Figures 23 and 24, 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 25a and 25b 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 second segment size with 80% traffic load, T_i with 100 ms is affected by the traffic so that the quality is improved. However, considering the 90% and 98% traffic loads, T_i with 1000 ms is also affected by the congestion. Moreover, the number of quality switches between representations is reduced when we use BFD as shown in Figures 26a and 26b. The number of quality switches is the lowest for the T_i with 100 ms regarding 10-sec and 1 second segment sizes.

7. Discussion

As we indicated in the Related Work section, studies on the fault tolerance problem in SDN have no specific explanations about how they create the failure in the network. This is crucial since applying a fault tolerance solution to the physical devices depends on the initiation of the failure. As shown in Table 2, the majority of studies in this area utilize Mininet as the emulation environment. However, as we describe the technical details in Section 5, causing a link failure in Mininet also means the destruction of the ports with which the link is associated. As a result, the switch is informed about the failure instantly which is not realistic. Therefore, the recovery time of the system cannot give correct results. For this reason, we use the Linux bridge in Mininet to initiate the link failure that is based on the broken link concept. In this case, the corresponding ports are not affected by the failure and continue to work as expected. Thus, the switch cannot perceive the failure immediately if an additional protocol is not used. Our results show that if traditional restoration applications are used in SDN without an additional protocol like BFD, they are unsuccessful in terms of recovery time. Since the Linux bridge can be used by any emulation/simulation environment that works on a Unix based system, our approach can be easily applied by future studies. Moreover, since this approach is more realistic, the results obtained by this method can be considered for real system implementation.

One of the important features of a fault tolerance scheme is the ability to handle multiple failures in terms of links and nodes (switches). Since we do not want to disrupt the reader from the context, we did not include the multiple failure case in this study. Our observations showed that our fault tolerance package in SDN has the capability to handle multiple failures. Note that a node failure can also be considered as a multiple link failure that is achieved by our schemes. Apart from our context, if we include the multiple link failure case, its performance evaluation may cause redundancy since actually the same action is performed for both single and multiple failures: finding a new route from one point to another.

Another aspect of our study is focusing on the quality of alternative paths. In the literature, even though many methods are used in order to reduce the recovery time and to eliminate the involvement of the controller, none of them consider the alternative path quality for fault tolerance in SDN. The quality of alternative paths is vital, since short recovery time is not beneficial when the quality of the affected flows decrease. To demonstrate its importance, we propose the DPQoAP module and compare the performance of it with the widely used protection and restoration approaches. Our results validate our proposition showing that QoS and QoE are extremely affected by the network condition if QoAPs are not considered. Even though the traditional protection approach provides low recovery time, the QoS and QoE would be unstable which may also invalidate SLA requirements of real-time flows. Thus, if our approach can be deployed in, possible network faults can be recovered without causing service quality losses for real-time applications like DASH. Also, considering the fact that video traffic is occupying 82% of the overall network traffic, we achieved notable success.

8. Conclusion

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 dynamic protection considering the quality of alternative paths for 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, DPQoAP. Based on the results of our experiments, we clearly state that dynamic protection for fault tolerance is crucial in terms of 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. 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 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 DPQoAP is not used, the recovery time would not be important since the QoE significantly deteriorates 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; 100 ms and 1000 ms BFD intervals; 40 Mbps, 45 Mbps, 49 Mbps traffic loads with the capacity of 50 Mbps. Our results showed that QoE parameters of the 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 100 ms is more sensitive to the traffic load, it detects congestion earlier than the 1000 ms interval so that the output of QoE parameters is better than the latter.

In the future, we plan to include the 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.

Acknowledgment

This work is supported by the Turkish Directorate of Strategy and Budget under the TAM Project number 2007K12-873, and the Galatasaray University Research Foundation under the Grant No. 18.401.003.

References

van Adrichem, N.L., van Asten, B.J., Kuipers, F.A., 2014. Fast Recovery in Software-Defined Networks, in: 2014 Third European Workshop on Software Defined Networks, IEEE. pp. 61–66. URL: http://ieeexplore.ieee.org/document/6984053/, doi:10.1109/EWSDN.2014.13.

Al-Tam, F., Correia, N., 2019. Fractional switch migration in multi-controller software-defined networking. Computer Networks 157, 1–10.

Bagci, K.T., Sahin, K.E., Tekalp, A.M., 2017. Compete or collaborate: Architectures for collaborative dash video over future networks. IEEE Transactions on Multimedia 19, 2152–2165.

Beheshti, N., Zhang, Y., 2012. Fast failover for control traffic in software-defined networks, in: 2012 IEEE Global Communications Conference (GLOBECOM), IEEE. pp. 2665–2670.

Bentaleb, A., Begen, A.C., Zimmermann, R., 2016. Sdndash: Improving qoe of http adaptive streaming using software defined networking, in: Proceedings of the 2016 ACM on Multimedia Conference, ACM. pp. 1296–1305.

Bentaleb, A., Begen, A.C., Zimmermann, R., Harous, S., 2017. Sdnhas: An sdn-enabled architecture to optimize qoe in http adaptive streaming. IEEE Transactions on Multimedia 19, 2136–2151.

Bhatia, J., Dave, R., Bhayani, H., Tanwar, S., Nayyar, A., 2020. Sdn-based real-time urban traffic analysis in vanet environment. Computer Communications 149, 162–175.

Bhatia, J., Kakadia, P., Bhavsar, M., Tanwar, S., 2019. Sdn-enabled network coding based secure data dissemination in vanet environment. IEEE Internet of Things Journal .

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

Cascone, C., Sanvito, D., Pollini, L., Capone, A., Sansò, B., 2017. Fast failure detection and recovery in sdn with stateful data plane. International Journal of Network Management 27.

Cheng, Z., Zhang, X., Li, Y., Yu, S., Lin, R., He, L., 2017. Congestion-aware local reroute for fast failure recovery in software-defined networks. IEEE/OSA Journal of Optical Communications and Networking 9, 934–944.

Cisco Visual Networking, 2020. The zettabyte era: Trends and analysis. Cisco white paper Accessed on: April 25, 2020. [Online]. Available: https://bit.ly/2S6luNL.

Correia, N., Faroq, A.T., 2019. Flow setup aware controller placement in distributed software-defined networking. IEEE Systems Journal .

dash.js, 2019, . DASH Industry Forum. URL: http://cdn.dashjs.org/latest/jsdoc/index.html. accessed at May 29, 2019.

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

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

Floodlight Controller, 2019, . Project Floodlight. URL: http://www.projectfloodlight.org/floodlight/. accessed at May 29, 2019.

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

Gozdecki, J., Jajszczyk, A., Stankiewicz, R., 2003. Quality of service terminology in ip networks. IEEE Communications Magazine 41, 153–159.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S., 2008. Nox: towards an operating system for networks. ACM SIGCOMM Computer Communication Review 38, 105–110.

Gyllstrom, D., Braga, N., Kurose, J., 2014. Recovery from link failures in a smart grid communication network using openflow, in: 2014 IEEE International Conference on Smart Grid Communications (SmartGridComm), IEEE. pp. 254–259.

iPerf, 2019, . iPerf Traffic Generator. URL: https://iperf.fr. accessed at May 29, 2019.

Katz, D., Ward, D., 2010. Bidirectional Forwarding Detection (BFD). RFC 5880. URL: https://rfc-editor.org/rfc/rfc5880.txt, doi:10.17487/RFC5880.

Kempf, J., Bellagamba, E., Kern, A., Jocha, D., Takács, A., Sköldström, P., 2012. Scalable fault management for openflow, in: Communications (ICC), 2012 IEEE international conference on, IEEE. pp. 6606–6610.

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

Kim, S., Son, J., Talukder, A., Hong, C.S., 2016. Congestion prevention mechanism based on q-leaning for efficient routing in sdn, in: Information Networking (ICOIN), 2016 International Conference on, IEEE. pp. 124–128.

Lee, K., Kim, M., Kim, H., Chwa, H.S., Lee, J., Shin, I., 2019. Fault-resilient real-time communication using software-defined networking, in: 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE. pp. 204–215.

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

Lu, Y., Zhu, S., 2015. Sdn-based tcp congestion control in data center networks, in: Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance, IEEE. pp. 1–7.

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

Nasimi, M., Habibi, M.A., Han, B., Schotten, H.D., 2018. Edge-assisted congestion control mechanism for 5g network using software-defined networking, in: 2018 15th International Symposium on Wireless Communication Systems (ISWCS), IEEE. pp. 1–5.

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

Open vSwitch, 2019, . Open Virtual Switch. URL: `http://www.openvswitch.org`. accessed at May 29, 2019.

Oyman, O., Singh, S., 2012. Quality of experience for http adaptive streaming services. IEEE Communications Magazine 50.

Petroulakis, N.E., Spanoudakis, G., Askoxylakis, I.G., 2017. Fault tolerance using an sdn pattern framework, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE. pp. 1–6.

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

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

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

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

da Rocha Fonseca, P.C., Mota, E.S., 2017. A survey on fault management in software-defined networks. IEEE Communications Surveys & Tutorials 19, 2284–2321.

Saraswat, S., Agarwal, V., Gupta, H.P., Mishra, R., Gupta, A., Dutta, T., 2019. Challenges and solutions in software defined networking: A survey. Journal of Network and Computer Applications 141, 23–58.

Seufert, M., Egger, S., Slanina, M., Zinner, T., Hossfeld, T., Tran-Gia, P., 2015. A survey on quality of experience of http adaptive streaming. IEEE Communications Surveys & Tutorials 17, 469–492.

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

Sharma, S., Staessens, D., Colle, D., Pickavet, M., Demeester, P., 2013a. Fast failure recovery for in-band openflow networks, in: Design of reliable communication networks (drcn), 2013 9th international conference on the, IEEE. pp. 52–59.

Sharma, S., Staessens, D., Colle, D., Pickavet, M., Demeester, P., 2013b. Openflow: Meeting carrier-grade recovery requirements. Computer Communications 36, 656–665.

Song, S., Park, H., Choi, B.Y., Choi, T., Zhu, H., 2017. Control path management framework for enhancing software-defined network (sdn) reliability. IEEE Transactions on Network and Service Management 14, 302–316.

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

Tajiki, M.M., Shojafar, M., Akbari, B., Salsano, S., Conti, M., Singhal, M., 2019. Joint failure recovery, fault prevention, and energy-efficient resource management for real-time sfc in fog-supported sdn. Computer Networks 162, 106850.

Thorat, P., Raza, S., Kim, D.S., Choo, H., 2017. Rapid recovery from link failures in software-defined networks. Journal of Communications and Networks 19, 648–665.

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

Varis, N., 2012. Anatomy of a linux bridge, in: Proceedings of Seminar on Network Protocols in Operating Systems, p. 58.

Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P., 2004. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing 13, 600–612.

Wireshark, 2019, . Wireshark Network Protocol Analyzer. URL: `https://www.wireshark.org`. accessed at May 29, 2019.

Yu, Y., Li, X., Leng, X., Song, L., Bu, K., Chen, Y., Yang, J., Zhang, L., Cheng, K., Xiao, X., 2018. Fault management in software-defined networking: A survey. IEEE Communications Surveys & Tutorials .

Yuan, B., Jin, H., Zou, D., Yang, L.T., Yu, S., 2018. A practical byzantine-based approach for faulty switch tolerance in software-defined networks. IEEE Transactions on Network and Service Management 15, 825–839.

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

Zhang, Y., Beheshti, N., Tatipamula, M., 2011. On resilience of split-architecture networks, in: 2011 IEEE Global Telecommunications Conference-GLOBECOM 2011, IEEE. pp. 1–6.

Zhu, S., Lan, S., 2015. Action based proactive node failure protection mechanism for openflow, in: 2015 IEEE International Conference on Progress in Informatics and Computing (PIC), IEEE. pp. 65–70.