Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research paper

Fault Tolerance of RTMP Protocol for Live Video Streaming Applications in Hybrid Software-Defined Networks

A. Oloomi, H. Khanmirza*

Computer Engineering Department, K. N. Toosi University of Technology, Tehran, Iran.

Article Info

Article History:
Received 14 August 2018
Reviewed 11 October 2018
Revised 04 March 2019
Accepted 11 May 2019

Keywords:
Software-defined network
Hybrid SDN
RTMP
Live video streaming
Fault Tolerance

*Corresponding Author's Email Address:
h.khanmirza@kntu.ac.ir

Abstract

Background and Objectives: Nowadays, video hosting services receive and stream videos using standard protocols like Real-Time Messaging Protocol (RTMP). During the streaming process, video file streams are usually divided into small multi-second parts, and the player receives these parts instead of the whole file at once. Most of the streaming protocols are capable of adaptive streaming and tolerating faults like device failures, and link disconnections. Faults might affect the performance of live streaming in terms of packet loss, latency, jitter, and video quality. The software-defined networking paradigm has also gained momentum in enterprise networks due to its lower-cost management and better network utilization. However, full migration from the current networks to the SDN model is not practical.
Methods: The purpose of this study is to investigate the effectiveness of fault tolerance mechanisms of RTMP protocol on hybrid software-defined networks (SDN). In this paper, a practical and straightforward hybrid network architecture is proposed for gradual migration from traditional IP networks. Then, the performance of the RTMP protocol is compared for live video streaming on this network with different streams facing multiple failures.
Results: Our experiments show that network failure recovery time in SDN is directly depends on the video stream recovery time while in traditional networks, streams need to be buffered again and it takes another several seconds due to the long interruption time. We propose an equation to give a rough estimation of data loss in SDN network during failures based on our observations which helps us in comparisons. We also demonstrate the average switching time in the SDN networks is almost half of the switching time in traditional networks.
Conclusion: Our experiments proves, practically, video recovery time in SDN is less than a traditional network and has more correspondence with mechanisms of RTMP.

Introduction

Software-Defined Network (SDN) denotes a specific approach in designing networks that try to separate the decision-oriented control plane functions from data plane functions that mainly deal with the forwarding of network packets. The main advantage of this separation is in this fact that the responsibility of the control plane can be assigned to a software or a set of software packages as the brain of the network known as the network controller. The network controller dynamically and centrally controls the behavior of the whole network based on the defined policies, which give great flexibility to network providers to rapidly change the policies and

introduce new services based on the customer needs. It also reduces network operation costs by reducing the number of administrators and eliminating human errors.

The core of the network controller package provides a set of Application Programming Interfaces (APIs) for developing various controlling software and translates the intended policies into commands which are installed in data plane devices through protocols like OpenFlow. SDN paradigm increases flexibility and eases the management and debugging of the network, and for these reasons, it is strongly considered as a proper alternative for traditional IP networks [1]. Nowadays, some IT-related businesses have implemented this type of network in their enterprise or Wide-Area Network (WAN) networks known as Software-Defined WAN (SDWAN) technology. It is predicted that 25% of WAN traffic would be for SDWAN by 2021 [2].

For SDN to become a prevalent paradigm, traditional devices of networks should be upgraded or replaced, which imposes a high cost to the companies. Besides, the migration process may cause prolonged periods of servicing interruptions. The gentler solution to this problem is to move gradually toward SDN, where SDN-capable devices are progressively added to the network. In such networks, known as hybrid SDN networks, both types of traditional and SDN-capable devices co-exist and function simultaneously. In most hybrid SDN networks, some modules are installed in SDN data path devices or in the controller that makes them capable of understanding the protocols of traditional networks and talking with traditional devices. Fig. 1 shows a typical hybrid SDN network model [3].

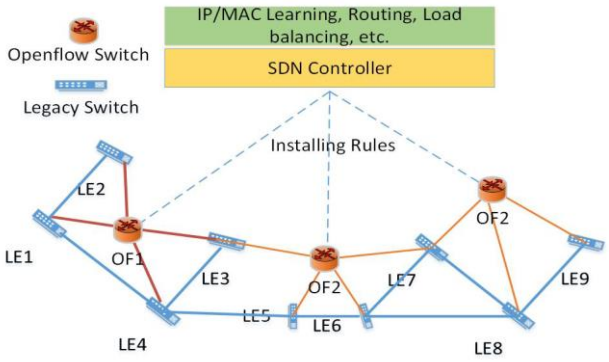

Fig. 1: Illustration of hybrid SDN network: SDN enabled OpenFlow switches operate alongside legacy switches [4].

Currently, video traffic accounts for more than 70% of Internet traffic, and it is likely to reach 90% by 2020. Real-time or live video streaming is of great importance in many applications, including live events, video conferences, video surveillance, and public safety. These applications need a reasonable delay and jitter support from the underlying network to deliver a smooth playback and the expected user experience in case of congestion and network outage, which is very common in networks. Most video-on-demand (VOD) applications and streaming protocols provide various built-in high-level fault tolerance mechanisms like video stream replication and client-side buffering [5] to tackle the issue. These mechanisms are often difficult to design, put pressure on streaming servers, and require tighter integration of streaming protocols with underlying forwarding protocols. In our research, we mainly want to find the level of correspondence between fault-tolerance mechanisms in streaming protocols like RTMP and fault tolerance mechanisms in hybrid SDNs, especially when the network faces faults.

Various protocols have been presented for live video streaming, which can be selected using the service type or software/hardware infrastructure. Many of the video protocols employ adaptive streaming such that audio and video contents of the network are adapted based on constraints of the network, and each one uses its specific methods to stream data. RTMP (Real-Time Messaging Protocol) [6] is developed by Macromedia and is one of the most widely used protocols for video streaming and VOD applications on the Internet. RTMP uses Transmission Control Protocol (TCP) and employs a combination of buffering and prefetching methods to implement fault tolerance [7]. Separation of control and data plane in SDN helps to provide better fault tolerance in networks because the controller can understand the semantics of packets arriving from various sources and occurred faults, and also it has a central view of all available resources [8]. Consequently, when a link is disconnected, as an example, the controller can understand which links are active and which available resources should be substituted to deliver the services without interruption. Without the dynamic programmability feature of SDN, providing a fault-tolerant service becomes a real challenge to the degree that most approaches on video streaming over SDN are compared with high capacity and rather expensive traditional network environments [9]. By the emergence of the SDN paradigm, valuable researches are done on video over SDN subject. Most of the researches focuses on traffic engineering and efficient routing algorithms to provide quality of service for video streams [1] [10][11][12]. Some works also propose a new architecture for control and data plane [13][14]. So far, a few works have studied the behavior of video streaming protocols and their fault tolerance mechanisms, especially in live stream scenarios. Authors in [5] have tested several streaming protocols on a pure SDN testbed. Most of the other works focus on tolerating faults in the control plane [15][16][17][18].

In this paper, at first, we propose a hybrid SDN

architecture with an ONOS (Open Network Operating System) controller [19]. This architecture is built based on a real-world traditional enterprise network. The new architecture helps current administrators with little knowledge of SDN to experience a new paradigm of networking and gradually shift their production network toward the full SDN model based on their budget and intended plan. In addition, our proposed approach does not force to employ additional and unnecessary protocols to avoid complexity in building, configuring, and maintaining. In contrast with other hybrid approaches [3] [20], our emphasis is on simplicity.

In the next step, we conduct multiple live video streaming experiments with RTMP over the new hybrid network in the presence of various network faults. With these tests, we investigate the practicability and effectiveness of our proposed architecture in one hand and the fault-tolerance of RTMP protocol on the other hand. Our experiments show that the hybrid scheme, not only helps gradual transition to the new networking paradigm but also hides some severe shortcomings of traditional networks, especially in a faulty and congested environment. Based on experiments, we conclude that the SDN paradigm has better correspondence with the streaming mechanisms of RTMP. To this end, video is transmitted from a live video source in the traditional network to video streaming servers in SDN and traditional networks. To the best of our knowledge, this is the first research that studies the performance of live video streaming on a hybrid SDN network. The rest of this paper is organized as follows: Section II studies various faults in the networks. Section III presents the implementation and analysis methods. Section IV presents the empirical results. Finally, the paper is concluded in Section V, and future suggestions are presented.

Network Faults

Different types of failures might occur in a network that can be separated based on where the fault has occurred.

A. Data Plane Faults

The primary duty of a data plane consisting of SDN-capable switches is forwarding packets based on the rules installed in their route table. The most common error at this plane is a link failure, which might occur due to cable failure [21]. Such faults might also be due to software bugs or incorrect configuration of the devices by operators [22]. In this study, failures of the data plane are implemented as device failure or communication link disconnection, and fault recovery is performed on other active links [26]. In order to recover from such faults, techniques like proactive forwarding and planned back-up paths are used [24].

B. Control Plane Faults

SDN controller defines rules based on network policies and installs them on the network devices in the data plane. Controller failure, application unavailability, API failure are among the faults which might occur in this plane. Such faults can be recovered by creating a cluster of mirrored controllers. If a fault occurs in one controller, one of the cluster members becomes active, and loss of network control and traffic is avoided [25]. Other methods try to keep an SDN network active without using OpenFlow protocol and can be used in scenarios when the communication of a device with the controller is disconnected. In such circumstances, SDN-capable devices switch to the standalone mode and operate as a traditional network device and keep the network working until the communication path is recovered [26].

Implementation Method

Fig. 2 shows the implemented experiment environment of this manuscript. The environment is implemented in Graphical Network Simulator-3 (GNS3) [27]. GNS3 is a network emulator software and is primarily used for accurate modeling and analysis of network protocols and networks with real-world device models. Our network consists of two SDN and two traditional networks. We use OpenvSwitch (OVS) switches v2.4.0 [28] in the data plane and ONOS v1.15.0 in the controller plane [19]. Today, there are plenty of choices for the controller from the centralized, multithreaded to the distributed controllers, which are extensively discussed in. Since we are planning to implement the proposed architecture in a production network, it is not reasonable to use centralized controllers due to future scalability issues. On the other hand, most of the distributed controllers are commercial and closed source and force the user to use vendor-specific devices, according to [1]. To avoid vendor-lockin problems, we investigated open source solutions. Among open source solutions, only ONOS and OpenDayLight (ODL) [29] controllers are used in the production environment and actively supported by the major players of the computer network field.

Several reasons motivated us to prefer ONOS to ODL:

1. ONOS provides a useful platform for developing network applications for application-specific scenarios, including custom communication routing, management, or service monitoring.
2. ONOS supports hardware and software updates without interrupting network traffic.
3. It can be executed as a distributed system with multiple servers and allows simultaneous use of their CPU and memory resources with failure tolerance
4. ONOS platform is developed as an extendible,

modular, and distributed controller.

5. ONOS has more flexibility in building the infrastructure layer such that it is able to configure OpenFlow switches to emulate devices of layer three and supports switching devices of layer 2 with OpenFlow capabilities.
6. It can install paths automatically using the Fast Reroute (FRR) mechanism, which significantly reduces switching time compared to a manual configuration.
7. ONOS requires less hardware compared to the ODL controller.
8. ONOS has a more robust cluster management system [30].

ONOS controllers are booted from the virtual environment running on Oracle VirtualBox [31] with identical hardware and are added to GNS3. Each of the controllers is executed on an Ubuntu virtual machine with a dual-core CPU, 2GB of RAM, 10GB of HDD, and a network interface.

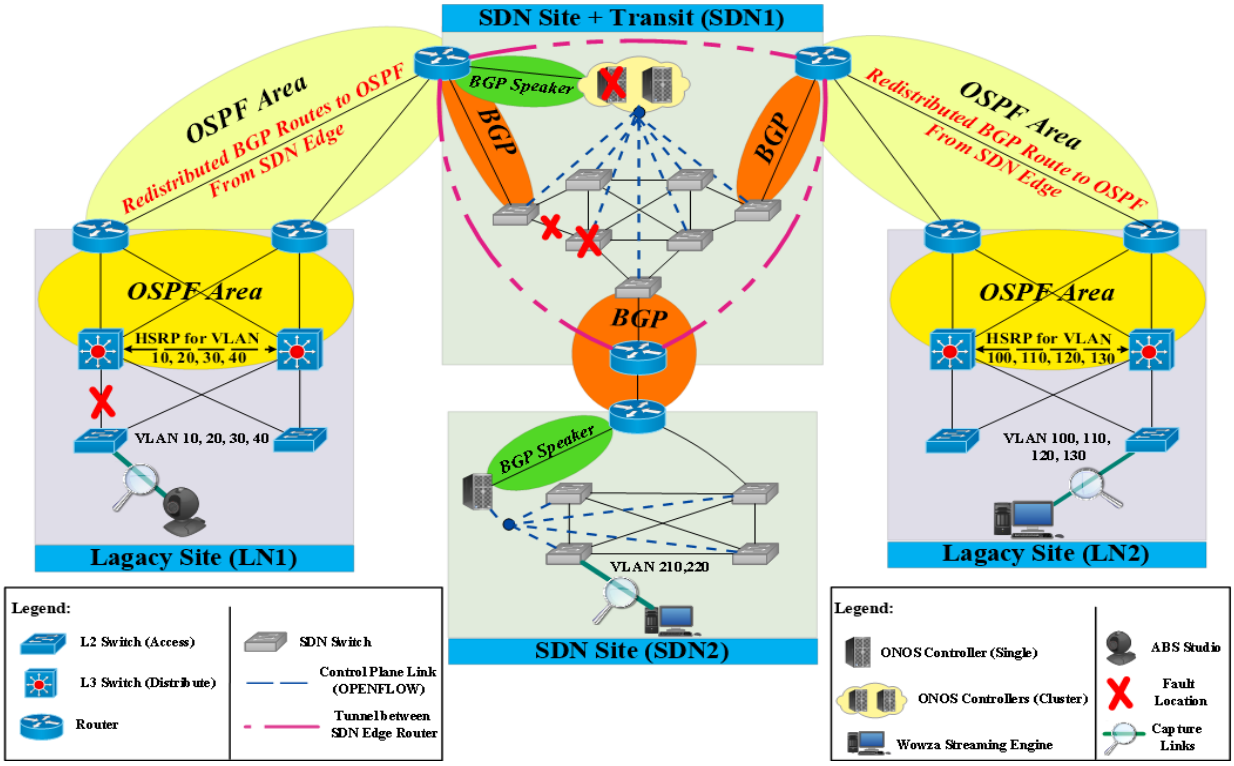

Fig. 2: Architecture of the implemented network platform. Red marks show the location of injected faults.

A. Architecture

In [3], several architectures listed to implement a hybrid SDN network. An organization might choose one of the architectures based on factors like its target budget for migration, current network design, and requirement of its services. As this paper is a research base for a real migration scenario, we selected an isolated island model, since it noticeably eases the migration process in several aspects. First, the organization can create a small SDN island according to its budget and gain real experiences by routing part of its actual traffic over the new network. Second, the island can be extended, duplicated, and tested without interfering with the old network. Third, the new islands can be used to build new service types or application domains in addition to the old services over the traditional network. To ease the development of the hybrid scheme, we created the SDN site, named SDN1, in the middle of the traditional network in a way that the network is partitioned into two isolated parts. With this configuration, the SDN site acts as a transit network and can be extended in all directions toward edges switch-by-switch until all the network turned into the full SDN model.

This architecture has an additional advantage that the SDN site manages the core of our network, where the transition is more straightforward due to having less protocol variety. However, in the core, the traffic is aggregated from several edge points and needs smarter management to meet service level agreements. SDN1 has a cluster of controllers, including two ONOS controllers, where their state is synchronized automatically and continuously.

The SDN2 site models a customer site and implemented to compare the SDN-to-SDN and SDN-to-traditional interactions in failures.

At the edge of both SDN networks, Cisco routers are used to communicate with the external network and

route the packets. In SDN2 and LN2, there is a video streaming server that uses the Wowza streaming server [32], streaming RTMP video traffic with different bit rates, frame rates, and resolutions. In LN1, using Open Broadcaster Software (OBS) studio [33], which is a live video stream recorder and player software, live video is captured and is transmitted to video streaming servers. In SDN1, seven OVS instances are installed and are connected to the controllers' cluster. With this strategy, both controllers in the cluster have full control over all of the switches, and if one of the controllers fails, the other controller takes control. In SDN2, there are five OVSes managed by a standalone controller. When the switches are connected to their respective controllers, rules of their flows are installed into ONOS.

Fig. 3 shows the resultant topology of SDN1 after the switches receive OpenFlow packets by the controller cluster. All controllers of the cluster are defined in each OVS instance. When the switch is turned on, and the OpenFlow packet is transmitted, the workload on each controller is reduced, and each controller manages the switches actively. Assigning switches to the controller in new versions of ONOS is performed by Atomix [30].

In experiments, we create intentional faults in SDN1 and LN1 networks. In traditional networks, we employ the recommended three-layer hierarchical design of Cisco [34], and it corresponds to our current network architecture. Both traditional networks use the Open Short Path First (OSPF) protocol for routing, and each of them is a separate subnetwork. In all experiments, clocks of all devices get synchronized. Hybrid networks must employ a dual routing system since traditional networks use distributed protocols to build routing tables, while in SDN networks, the controller centrally calculates and dictates the routes. In a common and straightforward method, the controller has the responsibility for connecting two sides by translating routing information. In this model, all SDN switches relay all protocol-related packets toward the controller. The controller understands these packets using a set of installed helper modules and reads and applies the routing information into the SDN data plane. It also distributes the routing information from the SDN side to the traditional side with the same protocol used on the traditional side.

In our proposed topology, a tunnel is created between edge routers of the SDN networks, and the tunnels are defined as OSPF interfaces. The routers at the edge of SDN1 receive traffic of OSPF and apply it to their routing table. Thus, the traffic of OSPF packets does not directly enter SDN1. If this mechanism is not used and the packets directly enter SDN1, since OSPF packets are unknown to the switches, they deliver the unknown traffic to the controller. In the controller, the OSPF routing module should be installed to handle these packets, which produces a high workload on the controller, as discussed. Routing in SDN can be executed either manually or dynamically using routing applications written on top of ONOS. Migrating from the traditional world and having a great experience, we decided to employ the Border Gateway Protocol (BGP) for routing inside SDN1 and SDN2 networks. ONOS has built-in support for BGP via a module. Switches in SDN network talk with the controller with BGP, and the controller builds the central and aggregated routing table with BGP advertisements and applies rules downward to switches with OpenFlow protocol. To connect devices of SDN and traditional network, edge routers of SDN distribute the routes inserted by BGP in their routing table in the OSPF.

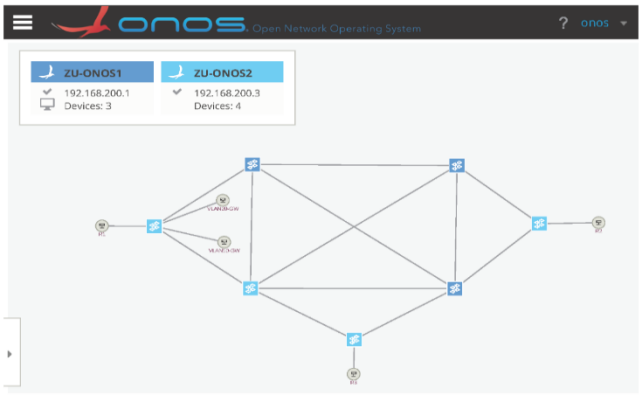

Fig. 3: Topology of SDN1 in ONOS.

Some hybrid approaches, such as SDN-IP architecture [20], suggest treating each SDN island as an AS (Autonomous System). They developed an application that provides the integration mechanism between BGP and ONOS within the AS. At the protocol level, SDN-IP behaves as a regular BGP speaker. Our architecture has several advantages over such proposals. First, such approaches impose the overhead of running another protocol (BGP) beside the intra-AS protocol. All switches in the traditional sites must understand BGP protocol in addition to OSPF, which complicates the configuration, operation, and maintenance of the network. Second, BGP is a complex internet-scale routing protocol and principally used in ISP or service provider networks, and basically, it is not intended to be used in organizations or enterprise networks. In the proposed approach, BGP is used only inside the SDN1 site, and LN networks are not aware of BGP as they talk with their neighbors with OSPF packets through tunnels.

B. Analysis

In the implemented network, the video stream is transmitted for live streaming to the Wowza Streaming Engine by OBS Studio, and Wireshark collects video traffic at the beginning and end of the network. With this method, the output traffic of OBS and input traffic to

Wowza can be easily captured for future analyses. To evaluate the performance of RTMP when a fault occurs, we tested videos of different qualities, as shown in Table 1. According to this table, there are low bit rate and high bit rate test for each video quality. Besides, video streaming for each bit rate is performed with two frame rates. For each experiment, data is collected with and without fault. According to the collected samples, the performance of RTMP is evaluated based on fault recovery time in SDN, video data loss, latency, and jitter. To measure the lost video data, we use the following formula to estimate the real data loss:

$$L_{est} = N_{avg} * R * S_{avg} \quad (1)$$

In this formula, N_{avg} is the average number of video packets in a second in the corresponding fault-less scenario, and R and S_{avg} are network recovery time and the average size of the packets, respectively. Using the above formula, we estimate the amount of lost data for each sample, and we report the average of high and low bitrates. We also calculate latency and jitter using the average Round Trip Time (RTT) reported by Wireshark.

Table 1: Video Streaming Parameters

Streaming Protocol	Resolution (pixel)	Bitrate (bps)	Framerate (fps)
RTMP	240	200 - 700	24 - 60
RTMP	360	400 - 900	24 - 60
RTMP	480	500 - 1000	24 - 60

C. Data Plane Fault Tolerance

To test data plane fault tolerance, we remove the links or switches between source and destination in the test environment, which forces the controller to re-calculate the new routes. This process must be done in a limited time period in a way that live streaming is sustained during the auto-recovery process without user intervention.

D. Control Plane Fault Tolerance

Control plane fault tolerance is defined as the ability to recover the control of the data plane after the failure of an SDN controller, and the data plane continues its operation. To test, the main controller in the cluster is disabled.

E. Fault Tolerance in Traditional Networks

In the design of traditional networks, having redundant links is an important feature that prevents interruption of traffic flows in case of link failures. However, Ethernet protocol with the MAC learning mechanism [35] does not support reserving redundant links as it can cause the creation of loops. Loops can paralyze a network, especially in broadcast scenarios. With loops in the network, broadcast traffic is continuously flowing and get larger in each hop and stops only when the bandwidth is filled, and devices halt due to massive traffic volume. In large LANs to have redundant links, Spanning Tree Protocol (STP) [36] is used, which is a protocol that blocks the redundant route when it is not required but allows us to have several reserved paths between switches; one route as the main route and others as reserves. If the main route is disconnected, the reserve route immediately replaces the main one. In order to compare the recovery time of the traditional network with SDN, link failure faults are applied in LN1.

Empirical and Analytical Results

A. Data Plane Fault

a) Estimating Video Data Loss

Fig. 4 shows the transmitted and received video data and estimation of lost data in a 1-min video stream in Kilobyte (KB). It is trivial that with a higher quality of videos, data loss increases because higher resolution and framerate increases the data rate. According to Fig. 4, the data loss in the best quality is about 2MB, which is approximately four times more than what the minimum quality misses. Considering the diagram represented in Fig. 5, the percent of loss in all videos is almost the same. This indicates that data loss is quite constant. In other words, if two to three frames are lost in low-quality video streaming, the same amount of data is lost in higher quality videos. Data loss in SDN is far less than a traditional network which is discussed in 4.c.(a)

The important point which can be seen in Fig. 5 is the accuracy of data loss estimation based on (1). By adding the amount of received data at the destination with the amount of estimated data loss, total estimated transmitted data volume is obtained, which slightly differs from the total measured transmitted data at the source (numbers written at the top of the columns). This difference mainly relates to the adaptive streaming mechanism of RTMP that adjusts the quality of video and amount of data based on the network condition. Considering these results, we can conclude that (1) has a close-to-reality estimation of data loss in short periods of streaming.

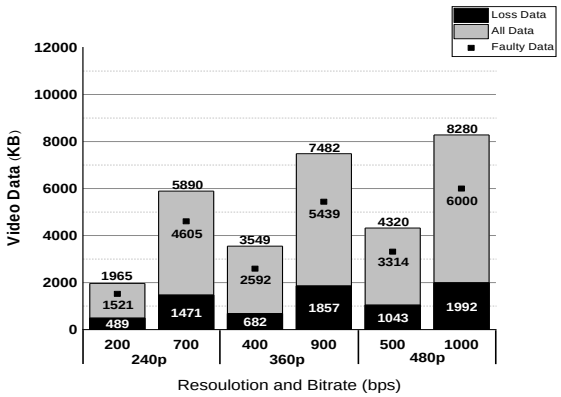

Fig. 4: Data Loss estimation.

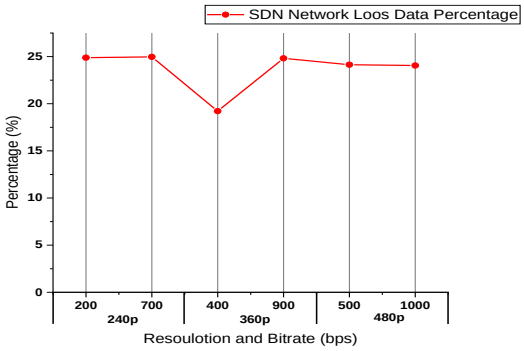

Fig. 5: Data Loss Percentage.

b) Latency

Fig. 6 shows the average latency with and without fault using the RTT parameter measured with Wireshark in milliseconds. Based on Fig. 6, there is no meaningful relationship between latency and video stream quality. However, occurring of faults in the network increases latencies up to 4 times. Amount of delays due to faults strongly depends on the controller speed to detect and recover the faulty route, which has a tight dependency with switching time and is discussed in 4.D(a). This result was expected due to this fact that increasing the framerate increases the amount of data injected into the network core, and this is not related to the delay of data. However, with higher frame rates, more packets experience higher delays.

c) Jitter

Fig. 7 shows the average jitter in the original network and the presence of faults. This parameter is calculated based on changes in latency between all consecutive packets. In general, jitter in both is negligible because the operational environment is a simulation environment. Similar to the latency, jitter increases with faults which were expected; since when data transmission is disrupted, latency in receiving the packets at the destination increases and causes significant gaps in latency. Also, like latency, this parameter does not depend on the quality and streaming rate of the videos, too. Jitter amplitude mainly depends on network condition, the number of streams, and their behavior.

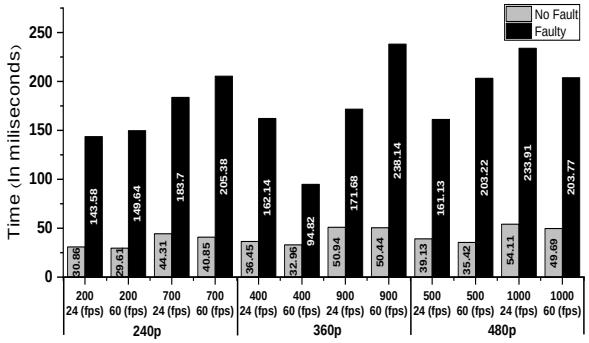

Fig. 6: Average Latency (ms).

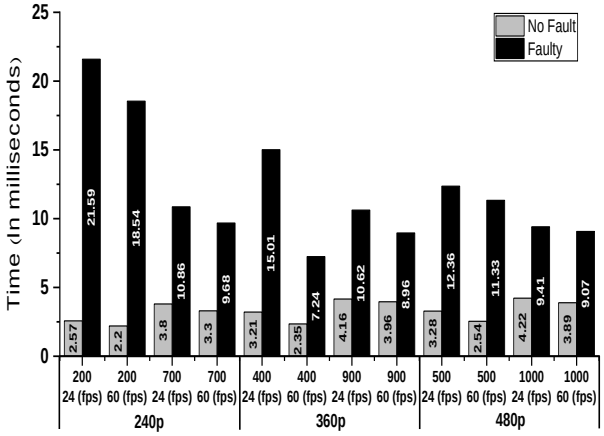

Fig. 7: Average jitter (ms).

d) Switching Time

Fig. 8 shows the switching time when a fault occurs in the network. These results indicate that traffic recovery time very closely matches the average switching time. Video streaming traffic is affected until the controller detects the fault and uses a redundant route. During this recovery process, the live video freezes on the Wowza stream engine.

In SDN1, the average fault recovery time in the data plane is around 15 seconds, with a maximum latency of 17 seconds, which is an acceptable time to recover from a disconnection fault. In all of the obtained samples, RTMP was able to recover the live video stream when a failure occurred.

B. Control Plane Fault

In order to test control plane failure, one of the ONOS controllers in the cluster is disabled. All OVS switches under control of the disabled controller continued their normal operation as the other controller instantly takes the control, and no video stream traffic is affected. This behavior was anticipated from ONOS, as recovering from this kind of fault is supported by default [37]. When both controllers of SDN1 are disabled, data plane traffic is affected, because there is no other controller to take control of OVSs.

For such conditions, ONOS pushes OVSs to a pre-defined standalone mode.

Indeed, when OVS instances get disconnected from the controller, they operate as a traditional switch and receive the traffic from the port learned through MAC learning until their connection to the controller is recovered. Another scenario of control plane failure is the disconnection of the communication cable of one of the ONOS controllers, which shows a behavior similar to the case in which one of the controllers is disabled, and the data plane was not affected. This case is verified by the Wowza server, using ping and collecting traffic by

Wireshark.

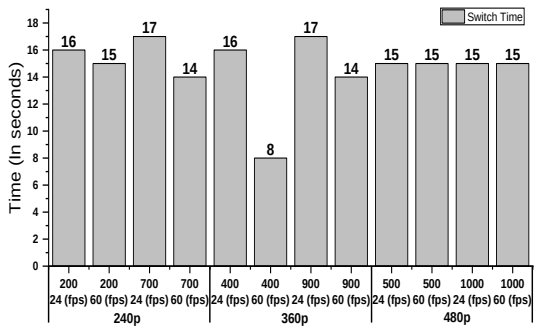

Fig. 8: Switching time of the Network for recovering video stream traffic.

Table 2: Comparison of data lost in SDN and traditional network Numbers calculated using (1) and results of Fig. 8 and Fig. 9

Resolution (pixel)	Bitrate (bps)	Total Data Sent (KB)	Loss Data in SDN (KB)	Loss Data in Legacy Network (KB)
240	200	1965	489	1515
240	700	5890	1471	4560
360	400	3549	682	2680
360	900	7482	1857	5697
480	500	4320	1043	3372
480	1000	8280	1992	6377

C. Fault in Traditional Network

a) Video Recovery Time

To compare the behavior of video recovery time in traditional networks with SDN, we disconnected a link in the LN1 site. After disconnection, STP enters the listening and learning steps after sensing the interruption and finally enters a forwarding state. This process takes 30 to 40 seconds from detection to retransmission. Since the processing time of STP is specified by the standard, video recovery time is calculated through collecting data by Wireshark, which is shown in Fig. 9. As can be seen, the average video recovery time at the destination is 48 seconds.

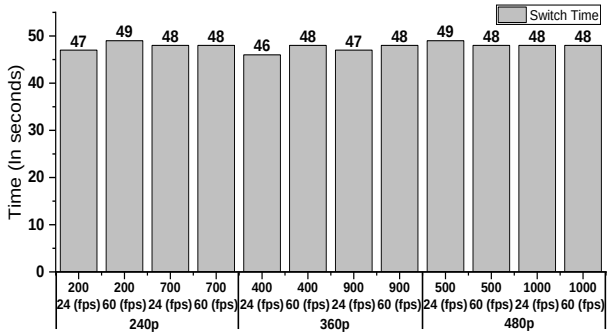

Fig. 9: Video stream traffic recovery time.

By substituting the recovery times obtained in this experiment and from the experiment 4.A.d in (1), amount of lost data when a fault occurs in the SDN and traditional network is obtained; for example, for the best quality scenario, 480p-1000, this value is 1992 KB in SDN while it is 6377 kB in a traditional network. The amount of loss for other scenarios is given in Table 2.

Fig. 10 compares the data loss percentage in SDN and traditional networks. As can be seen, data loss in the traditional network is about three times that of SDN.

We also observed an operational difference for RTMP in SDN and traditional networks in the presence of this fault. In the SDN test, RTMP protocol was interrupted around 15 seconds and recorded one integrated file while the video remained standstill for this period.

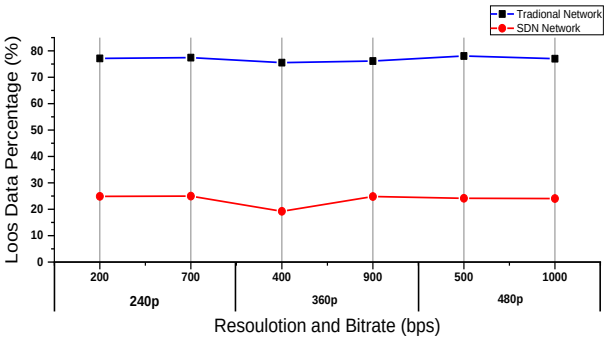

Fig. 10: Comparing the percentage of data loss in SDN and traditional network .

However, in experiments performed on the traditional network, which was interrupted around 40 seconds, the recorded file is divided into two sections. The recording was stopped after 25 seconds, and after the support link is installed and the communication link established with the server, capturing continued with a new file. We believe that this behavior relates to the RTMP standard, which requires some time for buffering after a long interruption. According to our measurements, the buffering time is about 10s.

Results and Discussion

Real-time video streaming is an essential solution for many applications like public safety, entertainment, and teleconferencing. However, there is a lack of understanding of how different real-time video streaming protocols perform in terms of fault tolerance. In this study, fault tolerance performance of a well-known real-time video streaming protocol, RTMP, is tested in a hybrid SDN network. Many of the streaming services like Netflix and YouTube provide fault tolerance between the client and the server; however, in this study, fault tolerance between the video source and the server is tested.

Conclusion

According to the results, it can be concluded that: 1) network failure recovery time (switching time) in SDN is the same as the video stream recovery time while in traditional networks, streams need to be buffered again

and it takes another 10 seconds due to the long interruption time. 2) Equation (1) gives a reasonable estimation of data loss, and the numbers are very close to real measurements. 3) Current video traffic flows without interruption even if all controllers fail, but new video streams need at least one active controller. 4) The average switching time in the SDN network was 15 seconds, which is almost half of the switching time in traditional networks. 5) Video recovery time in SDN is less than a traditional network and has more correspondence with mechanisms of RTMP. 6) According to previous researches, best latency and jitter for live video streaming are 146 and 28 ms [38], and average values we obtained in SDN networks under faulty conditions are 179 and 12ms, which indicates the flawless operation of RTMP in hybrid networks.

In the next step, we are trying to test the performance of other real-time video streaming protocols on our hybrid networks.

Author Contributions

A. Oloomi designed the experiments, collected the data, carried out the data analysis, and interpreted the results, wrote the manuscript in Persian. H. Khanmirza, the supervisor, translated the manuscript in English, Edited the primary and secondary research, and revise the paper.

Acknowledgment

The author gratefully acknowledges the H. Khanmirza for their work on the original version of this document.

Conflict of Interest

The author declares that there is no conflict of interest regarding the publication of this manuscript. In addition, the ethical issues, including plagiarism, informed consent, misconduct, data fabrication and/or falsification, double publication and/or submission, and redundancy have been completely observed by the authors.

Abbreviations

SDN	Software-Defined Network
API	Application Programming Interfaces
WAN	Wide-Area Network
SDWAN	Software-Defined WAN
VOD	Video-on-demand
RTMP	Real-Time Messaging Protocol
TCP	Transmission Control Protocol
ONOS	Open Network Operating System
GNS3	Graphical Network Simulator-3
OVS	OpenvSwitch
ODL	OpenDayLight
FRR	Fast Reroute
OBS	Open Broadcaster Software
OSPF	Open Short Path First
BGP	Border Gateway Protocol
N_{avg}	Average video packets in a second
R	Network recovery time
S_{avg}	Average size of the packets
RTT	Round Trip Time
STP	Spanning Tree Protocol
KB	Kilobyte
MB	Megabyte
ms	miliseconds
bps	bit per second
p	pixel

References

[1] D. Kreutz, F. M. V. Ramos, P. Esteves Verı´ssimo, C. Esteve Rothenberg, S. Azodolmolky, S. Uhlig, "Software-defined Networking: A Comprehensive Survey," Proceedings of the IEEE, 103(1): 10-76, 2015.

[2] Cisco Public, "The Zettabyte Era: Trends and Analysis," Cisco, 2017.

[3] Sandhya, Y. Sinha, K. Haribabu, "A survey: Hybrid SDN," in Journal of Network and Computer Applications, 100: 35-55, 2017.

[4] Rashid Amin, Martin Reisslein, Nadir Shah, "Hybrid SDN Networks: A Survey of Existing Approaches," in IEEE Communications Surveys & Tutorials, 20: 3259-3306, 2018.

[5] S. Gaikwad, S. Tafleen, R. Gottumukkala, K. Elgazzar, "Fault Tolerance of Real-time Video Streaming Protocols over SDN Networks," in Proc. 14th Wireless Communications and Mobile Computing Conference, IWCMC, Limassol: 101-107, 2018.

[6] H. Parmar, M. Thornburgh, "Adobe's Real Time Messaging Protocol," 2012.

[7] X. Lei, X. Jiang, C. Wang, "Design and implementation of streaming media processing software based on RTMP," in Proc. 5th International Congress on Image and Signal Processing, Chongqing: 192-196, 2012.

[8] H. Kim, N. Feamster, "Improving network management with software defined networking," IEEE Communications Magazine, 51(2): 114-119, 2013.

[9] Q. Wang, K. Xu, R. Izard, B. Kribbs, J. Porter, K.-C. Wang, A. Prakash, P. Ramanathan, "GENI Cinema: An SDN-Assisted Scalable Live Video Streaming Service," in Proc. IEEE 22nd International Conference on Network Protocols, Raleigh, NC: 529-532, 2014.

[10] M. Karakus, A. Durresi, "Quality of Service (QoS) in Software Defined Networking (SDN): A survey," in Journal of Network and Computer Applications, 80: 200-218, 2017.

[11] A. Ben Letaifa, "Adaptive QoE monitoring architecture in SDN networks: Video streaming services case," in Proc. 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 2017.

[12] Eirini Liotou, Konstantinos Samdanis, Nikos Passas, Lazaros Merakos, "QoE-SDN APP: A Rate-guided QoE-aware SDN-APP for HTTP Adaptive Video Streaming," in IEEE Journal on Selected Areas in Communications, 36(3): 598-615, 2018.

[13] H. Owens II, A. Durresi, "Video over Software-Defined Networking (VSDN)," in Computer Networks, 92(2): 341-356, 2015.

[14] Sahil Garg, Kuljeet Kaur, Neeraj Kumar, Joel J. P. C. Rodrigues, "Hybrid Deep-Learning-Based Anomaly Detection Scheme for Suspicious Flow Detection in SDN: A Social Multimedia Perspective," in IEEE Transactions on Multimedia, 21(3): 566-578, 2019.

[15] N. Katta, H. Zhang, M. Freedman, J. Rexford, "Ravana: controller fault-tolerance in software-defined networking," in Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined

Networking Research, New York: 1-12, 2015.

[16] C. Scott, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai, E. Huang, Z. Liu, A. El-Hassany, S. Whitlock, H. Acharya, K. Zarifis, S. Shenker, "Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences," in ACM SIGCOMM Computer Communication Review, 44(4): 395-406, 2014.

[17] T. Koponen, M. Casado, N. Gude, J. Stribling, "Onix: A Distributed Control Platform for Large-scale Production Networks," in Proc. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Vancouver: 351-364, 2010.

[18] B. Chandrasekaran, B. Tschaen, T. Benson, "Isolating and Tolerating SDN Application Failures with LegoSDN," in Proceedings of the Symposium on SDN Research, Santa Clara: 1-12, 2016.

[19] "Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions".

[20] "SDN-IP Architecture,"

[21] H. Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner and N. Feamster, "CORONET: Fault tolerance for Software Defined Networks," in Proc. 20th IEEE International Conference on Network Protocols (ICNP), Austin: 1-2, 2012.

[22] L. J. Jagadeesan, V. Mendiratta, "Programming the Network: Application Software Faults in Software-Defined Networks," in Proc. IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Ottawa): 125-131, 2016.

[23] B. J. v. Asten, N. L. M. v. Adrichem, F. A. Kuipers, "Scalability and Resilience of Software-Defined Networking: An Overview," in Networking and Internet Architecture, New York, 2014.

[24] S. Zhang, Y. Wang, Q. He, J. Yu, S. Guo, "Backup-resource based failure recovery approach in SDN data plane," in Proc. 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), Kanazawa: 1-6, 2016.

[25] A. Xie, X. Wang, W. Wang, S. Lu, "Designing a disaster-resilient network with software defined networking," in Proc. IEEE 22nd International Symposium of Quality of Service (IWQoS), Hong Kong: 135-140, 2014.

[26] "Open vSwitch Manual," Open vSwitch.

[27] "The official guide and reference for GNS3," GNS3, 15 January 2020.

[28] "Open vSwitch Downloads Page," Open vSwitch.

[29] "OpenDaylight SDN Controller,".

[30] "ONOS Wiki," ONOS.

[31] "Oracle VM VirtualBox," ORACLE.

[32] "Wowza Media Systems | Live Video Streaming Solutions,".

[33] "OBS: Open Broadcaster Software,".

[34] A. Bruno, S. Jordan, CCDA 200-310, CiscoPress,): 112-121, 2016.

[35] "Ethernet Protocol," ScienceDirect..

[36] "Spanning Tree Protocol (STP) Application of the Inter-Chassis Communication Protocol (ICCP)," IETF, 2016.

[37] A. S. Muqaddas, A. Bianco, P. Giaccone, G. Maier, "Inter-controller Traffic in ONOS Clusters for SDN Networks," in Proc. IEEE International Conference on Communications (ICC), Kuala Lumpur: 1-6, 2016.

[38] A. Nurrohman, M. Abdurohman, "High Performance Streaming Based on H264 and Real Time Messaging Protocol (RTMP)," in Proc. 6th International Conference on Information and Communication Technology (ICoICT), Bandung: 174-177, 2018.

Biographies

The Authors' photographs and biographies not available at the time of publication.

How to cite this paper:

A. Oloomi, H. Khanmirza, "Fault Tolerance of RTMP Protocol for Live Video Streaming Applications in Hybrid Software-Defined Networks," Journal of Electrical and Computer Engineering Innovations, 7(2): 241-250, 2019.

DOI: 10.22061/JECEI.2020.6416.314

URL: http://jecei.sru.ac.ir/article_1227.html