Int. J. Nonlinear Anal. Appl. 16 (2025) 5, 69–76
ISSN: 2008-6822 (electronic)
http://dx.doi.org/10.22075/ijnaa.2023.31916.4733

A new approach in fault tolerance in control level of SDN

Yasser Narimani[a], Esmaeil Zeinali[b,*], Abbas Mirzaei[a]

[a]*Department of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran*

[b]*Department of Computer Engineering, Ardabil Branch, Islamic Azad University, Ardabil, Iran*

(Communicated by Seyed Hossein Siadati)

Abstract

Nowadays, the world of communication and information has undergone many changes, the rapidly growing trend of computer systems in terms of computing and storage and retrieval methods on the one hand and on the other hand a slow or almost constant trend in computer networks and non-response to The current needs of companies, organizations, telecommunication service providers as well as network designers face a series of limitations such as complexity, lack of scalability and lack of coordination between market needs and network capabilities and the introduction of cloud services and virtual servers. Network designers have taken a big step forward in the development of networks, and the solution to achieve this progress in networks is the use of software-based networks. In recent years, despite much research in the field of this type of network, there is still a need for optimizations in this area. One of the challenges that currently needs work and development is the issue of fault tolerance and failure in these networks. In this paper, various failure tolerance methods in software-defined networks are introduced.

Keywords: Software defined networks (SDN), Network control level, Network data level, Fault tolerance
2020 MSC: 68M10, 93B70

1 Introduction

Software-defined networks are an evaluation of computer networks [14]. In traditional networks, when a package arrives at a switch, the software on the switch decides where the package should be sent [8]. The switch treats all packets in the same way and sends them in the same way [5]. SDN actually means the physical separation of the network control surface from the data surface, which controls the control of several network devices [12]. SDN is a new architecture that makes networks dynamic, manageable, cost-effective and flexible. Such features are ideal for today's applications, which inherently require high bandwidth and are dynamic. By separating the network control from the routing, this architecture allows the network control to be directly programmable and the network infrastructure to be abstracted for network applications and services [15]. These emerging networks can automate many network activities and can also develop and expand parts of the network in a completely software way. Due to some features of these networks, the probability of errors in these networks is high. One of the research topics is to investigate fault detection and increase fault tolerance in this type of network. It can be said that fault tolerance is one of the most important challenges in these networks [11]. In recent years, extensive research has been conducted to increase failure tolerances

*Corresponding author
Email addresses: yasser.narimani@gmail.com (Yasser Narimani), es.zeinali1973@gmail.com (Esmaeil Zeinali), mirzaei.a@gmail.com (Abbas Mirzaei)

Received: August 2023 *Accepted:* October 2023

in software networks. In the continuation of this article, in the second part, the work done in connection with this issue will be reviewed and the advantages and disadvantages of each will be described in a tabular form.

SDN is now known as a novel paradigm that promises better flexibility, programmability, and innovation in networks. Nonetheless, given the urgent requirements of the new generation of SDN, operators should evaluate the possible faults and threats accurately and consider effective solutions before the implementation phase. In this regard, availability and fault tolerance are among the major criteria that should be evaluated. Several papers have analyzed this problem in two different areas of fault tolerance: 1) data level and 2) control level.

The fault tolerance of the data plane includes such proposals as [1], where a mechanism to obtain path failure recovery times below 50 milliseconds is presented. The work presented in [6] is also focused on the data plane by providing fast-failure mechanisms to react to link or switch failures. On the other hand, the basic SDN architecture depicts the control plane as a potential single point of failure. Therefore, the design of a fault-tolerant control plane is a must. The straightforward solution is to have redundant controllers that can take responsibility in the case of a failure. However, as Section II-B will explain, the mechanisms used to implement such a solution are complex and pose challenges that cannot be neglected.

This paper is focused on the fault tolerance of the control plane with the assumption that consistency and performance are important criteria and must be considered. We define the performance of a controller according to the following two concepts: 1) Controller Latency: The time that an incoming request has to spend waiting once it is delivered at the ingress switch until the respective new data plane rules are established in the respective switches. 2) Controller Throughput: The maximum number of new flows handled within a time unit. In addition, we define consistency as the ability to have exactly the same view of a network at all controllers, which is an exact mapping of the network state, at any time.

2 Related works

Gonzalez et al [2] presented a method for Compatibility and fault tolerance in SDN controllers that have been considered and a new mechanism has been designed. The purpose of this study is to bring the performance of the SDN Master-Slave controller closer to the control provided by a single controller, regardless of the controller's failure. In this method, a simple duplication scheme is introduced that performs the correction check operation so that it affects the network performance in only a few gaps. To offer competitive performance, the Master-Slave controller requires a very reliable communication channel between the main controller and the database. It can be said that one of the most important criteria for providers when implementing a control system is fault tolerance. Therefore, the author assumes that this is necessary regardless of the error control mechanism.

Song et al [16] presented a method that in which, SDN reliability is studied from the perspective of the control path. Their focus is mainly on the level of reliability challenges at the control path between the control layer and the data layer. In this method, control path algorithms and a new control message classification and prioritization system are designed to increase SDN reliability. The issue of control path reliability between controllers and switches for SDN operations is inevitable. In the proposed method, the control path is considered as a network and several practical problems of SDN reliability are studied. Finally, effective solutions are provided to create a control path management reliability framework that includes plans for 1) pairing logical and physical redundancies, 2) controller cluster virtualization, 3) rapid and accurate fault detection and recovery, and 4) Message setting control to improve SDN scalability and reliability. Song et al. Measured the effectiveness of their proposals through comprehensive experiments in a real-world network system and implemented them using numerical analysis.

Hu et al. [4] presented a method and stated, Efficient and flexible link fault tolerance (FTLink) is a link failure and error management plan for SDN. This set creates backup links for the master link. In this method, a matching table is created to maintain the rule of backup through the controller. When the system detects a link failure, FTLink activates the backup link as the new master link for the failed link. The flow rules are installed on the switch using the controller after they have been matched to the created table. However, in this method, only a single control system is used and the controller failure events are not considered here. The results show that compared to the baseline designs, FTLink achieves TCAM with high-performance bandwidth and acceptable recovery time.

Li et al. [7] presented a method and stated that BOND is a flexible SDN network recovery system that deals with the link breakdown aspect of the network system. This method stores the backup rules on the switch to provide an alternative path in case the link fails. Also, a hash table is used to quickly recover a failed link. However, because SDN switches do not have resource constraints and computing in SDN, storing critical information such as routing paths in switches can affect system performance. In addition, if the switch does not work, the system loses all network routing information. In this method, a test bed is created to demonstrate BOND performance in the real environment. BOND performs as much as the pure failure protection scheme, which is expected. As a future work, this paper discusses the intention to expand BOND to support more advanced network flow needs.

Lee et al. [6] presented a method with the main focus on real-time flows, by considering the limitations of fault tolerance and using adaptive path repair. They implemented a multi-range routing algorithm that can redirect streams based on their time budget. The method proposed by Lee et al. Is FR-SDN, which supports fast and predictable path repair to meet real-time fault tolerance limits. This method has been introduced as the first research in the field of repair of adaptive paths in SDN networks, which aims to safely recover faults from link failures without violating any losses. The FR-SDN works in such a way as to change the path repair from the SDN control to the switch mode and make the path repair operation possible with minimal delay and limitation. FR-SDN, in addition to providing a feasibility study to estimate corporate firm flow losses in real time, also uses an adaptive MCP technique. With this technique, only some of the flows are selectively redirected to recalculate the route effectively, in a limited time budget. recalculation of the route is performed effectively. The FR-SDN prototype, such as a 1/10 scaled autonomous vehicle, is designed to evaluate its effectiveness in a real-time system. Evaluations show that FR-SDN significantly increases the reliability of real-time communications.

Akanbi et al. [1] presented a method and Stated that Fast Fail-Over Technique for Distributed Controller (FFDC) is a control panel management architecture distributed in SDN. This architecture uses a transfer information table (FIT) to update the failure event table. To reduce latency for existing streams, a copy of the request is stored by the FFDC, which is used in subsequent streams to reduce latency in the system. The mechanism of action is that the FFDC uses the broken links to determine the status of the controller, that is, if all existing links fail, the FFDC works on the basis that the controller is also damaged.

Mantas et al. [10] presented a protocol called Rama, and Rama's basis was to prevent a breakdown point (SPOF) at the control level without modifying the protocol or switch, which is contrary to Ravana [5102]. Its work is based on modifying the protocol. The Rama architecture is based on the Master-Slave approach, in which the master is the master controller and the copies are slave controllers. When the master controller is not working, the Rama, by selecting the slave controller, causes it to act as a main controller by using the coordinator service. However, the overhead in Ravana is less than in Rama. In Rama, there is no need to change the OpenFlow protocol or switches. This increases the cost because the techniques used in Rama are more expensive than Ravana. Since the overhead causes a reduction in performance, Mantas in practice expects this to be offset by the fact that their solution can be used immediately.

Hu et al [3] presented a method and expressed that Dynamic Slave Control (DSCA) allocation is another master-slave network architecture that can shift the switch from one failed controller to another active controller. This architecture is actually a distributed architecture with several controllers that affect the failure of the controller chain. When a series of controllers fail, the switch is transmitted to the slave controller. The basis of the work is that the backups or slave controllers are determined based on the existing load and the active steps that exist. Experimental analysis shows that DSCA can reduce latency in the worst case under controller failure. It should be noted that in this paper, failure time and power analysis using this mechanism are not provided.

Ali Malik et al. [9] presented a method and stated that smart routing, as an active convergence scheme, can predict link failure and formulate intermittent routing for the same link. This method can eliminate the risky links that lead to the loss of packets during transmission and increase access to network services. In this method, to evaluate the availability and reliability of network entities, two parameters are used, namely, the average time between failures and the average recovery time. Also, a single controller with an active error framework is used to manage the entire network system. Experimental findings clearly show that the proposed method is highly effective in increasing the availability of SDN services. Unfortunately, flap rates that can lead to failure predictions can lead to network instability, especially when reached at high rates. For this purpose, in this method, the percentage of unnecessary routing flaps was measured, and in the worst case, it was 25%, which is almost logical in practice. Smart routing does not include delays, feedback, or power assessments, but rather, the experimental analysis includes routing flaps and service availability.

Narimani et al. [13] introduced a dynamic architecture that focuses on the end-to-end mobility support required to maintain service continuity and quality of service. This paper also presents an SDN control plane with a stochastic network calculus (SNC) framework to control MEC data flows. Following the entrance processes of different QoS-class data flows, closed-form problems were formulated to determine the correlation between resource utilization and the violation probability of each data flow. Compared to other solutions investigated in the literature, the proposed approach exhibits a significant increase in the throughput distributed over the active links of mobile edge hosts. It

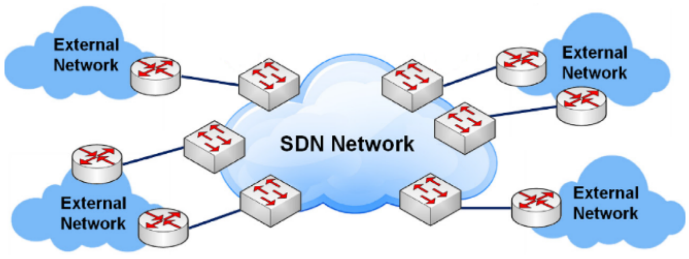

Figure 1: The architecture of SDN controller.

also proved that the outage index and the system's aggregate data rate can be effectively improved by up to 32%.

3 System model

Maintaining a consistent view of a network among all controllers is challenging. The existing mechanisms used for this purpose tend to affect negatively the performance of the controller platform. Thus, having mechanisms that achieve fault tolerance and consistency by keeping the performance consequences low is a clear goal in the research community. In this context, the main questions addressed by this paper are: How to bring the performance of a fault-tolerant controller close to that of a non-redundant (single) controller?

A fault-tolerant controller platform may be implemented by distributing the load among separate controller units, which means that the responsibility has to be spread over several domains [16]. On the other hand, there are simpler approaches known as Master-Slave, where a single controller is in charge of all decisions. It is supported by backup controllers having a synchronized view of the network, which can take responsibility for future decisions in the case of a failure. This paper is focused on Master-Slave scenarios since we believe that due to their simpler structure, they offer a better possibility to improve the trade-off between consistency and performance.

Before a master-slave controller is presented, this section discusses all the concepts required to develop SDN-oriented control networks. For this purpose, the main features and challenges of a single SDN controller are analyzed. After that, the previous studies on the design of the fault tolerance control level are reviewed.

According to the existing algorithms, the following actions are taken in a single SDN controller when a new data flow enters an SDN switch without specific forward or transmission instructions:

i. A small packet that indicates a new flow is received in the input switch of the network and then sent to the controller.
ii. Upon the flow request, network policies, and current network view, the controller calculates the forwarding path.
iii. The controller updates the relevant switches by adding an entry to the flow tables.

These three steps are taken into account to send all the next packets of the new flow based on the sent decisions calculated previously; therefore, there is no need for any actions from the control layer. Figure 1 demonstrates these steps. This is the simplest and most common method in which the new flows can operate in a software-defined network. Since the implementation of the first SDN samples, researchers have grown really fond of modelling and analyzing the controller performance with the motivation for a large-scale installation. For instance, the researchers in [11] and [4] evaluated the performance of a controller architecture under specific parameters to predict the probabilistic execution problems. According to their results, the SDN performance depends greatly on the processing power and speed of a controller. If the processing speed of a controller is not high enough, the network will considerably face difficulty in managing new flows, something which directly affects the QoS.

Despite the potential limitations and outcomes, the model proposed in this section (Figure 1) is the simplest and best SDN architecture. Thus, the main advantage of this paper is the development of a consistent fault-tolerant master-slave controller in a bid to achieve the closest performance to that of the proposed controller.

Based on the review of references regarding fault-tolerant SDN controllers, master-slave controllers were introduced as a simple subclass addressing this problem. In these controllers, a central unit (i.e., the master) is responsible for making urgent decisions at the data level. Moreover, a simple MasterSlave technique was proposed in [5], where active replication and passive replication methods were proposed for fault tolerance with a high accuracy. Nevertheless, this approach does not consider the problems of consistency. By contrast, the papers reviewed by [8] and [7] analyzed the master-slave architecture by considering the consistency problem. Both of these papers proposed an architecture using a joint database above the controller to retain a globally common NIB. Consistency is maintained as every change is replicated in the NIB; however, this can have a significant effect on the controller performance concerning the overhead time generated in each replication. Eventually, Ravana's recent study in [2] proposed an interactive fault-tolerant master-slave controller that would process the received messages in exchange and only once in a real-time framework. Ravana implements the SMR methods through ZooKeeper but increases the domain by executing the switch-side mechanisms to guarantee operation accuracy. Since this method is novel and reliable, some of its approaches were used in the proposed scheme. Nonetheless, some changes are proposed to eliminate the challenges introduced henceforth.

It is necessary to consider that an SDN controller should be fault-tolerant and always available. In virtual environments, both active replication and passive replication are two common techniques for achieving these goals [12]. Since active replication is a solution that provides high flexibility and slight interruption time, it can be used as an alternative or the main option [5]. A visual solution is to replicate all messages sent by a switch so that both controllers can receive the same copy and perform the same operations.

3.1 A. Performance of The Single SDN Controller

In a single SDN controller, when a new flow with no specified forwarding instructions comes into an SDN switch, the following actions are performed:

i. A packet representing a new flow is received at a network ingress switch and it is sent to the controller.
ii. The controller computes the forwarding path, depending on the flow request, the network policies and its current network view.
iii. The controller updates the respective switches by sending entries to be added to the flow tables.

After these steps, all subsequent packets of the new flow are forwarded, based on the pre-calculated forwarding decisions and do not need any control plane action. The mentioned steps are presented in Figure 1, and it is the simplest and conventional way to operate new flows in an SDN network. Since the launch of the first SDN solutions, there has been a huge interest in modeling and analyzing the controller performance, motivated by the requirements of a large-scale deployment. For instance, [11] and [4] evaluate how a single controller architecture will perform under certain parameters, in order to foresee potential implementation issues. Their results show that the performance of an SDN network has a strong dependency on the processing speed of the controller. If it is not fast enough, the capability of the network to handle new flows is limited considerably, affecting the overall user experience.

In spite of its potential limitations and consequences, the model presented in this section (Figure 1) is the simplest and best performing SDN architecture. Therefore, the main point of this paper is to develop a consistent and fault-tolerant Master-Slave controller, with performance as close as possible to the one offered by a single controller.

3.2 B. Fault Tolerant in Controllers of SDN

A single controller is a single point of failure, and hence unacceptable. Having a fault-tolerant SDN controller is a well-identified need addressed in several previous works. Among them, one of the first and most relevant proposals is ONIX [16]. They elaborate on the main concepts and pillars to be considered in the implementation of a general fault-tolerant controller platform, which has been used as a reference in most of the further related works. ONIX also provides a general framework where several important open issues have been identified, such as the trade-off between consistency, durability, and scalability. Although the specific solution remains open and up to the specific implementation needs. One of the most important concepts that our paper takes from ONIX is the Network Information Base NIB. They defined the NIB as a copy of the current network state which contains relevant information from all network entities within the topology. We follow the same definition, and from now on, we assume the NIB as all information that the control plane must know to perform network operations, modifications and all related decisions. Hyperflow [10] is another widely used fault-tolerant controller proposal. It is a distributed event-based control plane for OpenFlow that is logically centralized but physically distributed. The platform localizes decision-making to individual controllers, ensuring that all the instances are synchronized, considering inconsistency problems due to the synchronization speed.

The SDN controller must be fault-tolerant and highly available. In virtualized environments, active and passive replication are two common techniques used to achieve this [12]. Since active replication is a solution that offers high resilience and negligible downtime, its use may be seen as the primary alternative [5]. One intuitive solution is to duplicate all the messages sent by a switch, with the intention that both controllers receive a copy and perform identical operations. This approach may easily create inconsistencies between the network image of the controllers due to the reordering of events generated by delay differences, as Figure 2 illustrates. Even if we were able to enforce a strict order, non-determinism in the processing poses an additional huge challenge. This is the root problem of the performance/consistency balance, and it leads to the definition of the first challenge:

Challenge 1:. Avoid a single point of failure, considering synchronization problems between replicas caused by non-deterministic delays and processing.

Challenge 2:. Implement a simpler and less performance-expensive Master-Slave platform that guarantees NIB consistency.

Challenge 3:. Find which approach is better. Having replication of NIB updates, or having replication of incoming messages.

4 Performance evaluation

In this section, we evaluate the performance of the proposed controller and compare it with a single controller and other master-slave approaches. To do this, we use discrete event simulation.

In all analyzed controllers (unit controller, proposed solution and other Master-Slave controllers), the received message processing stage is common and this is the main process that affects the performance of all controllers. Here, we assume this process as a stochastic process with expected duration μ_M. Our proposed Master-Slave simulation follows all the concepts previously presented. Finally, since we lack specific technical details to make an exact comparison under similar conditions with other Master-Slave controllers, we use the following method. The main difference in our proposal is that any new stream arriving at the controller can be attended directly, as the time used for the buffering routine is the only common overhead. Compatibility check routines may be a significant overhead, but they are not applied to every controller operation. On the other hand, other Master-Slave approaches impose additional overhead on each input request due to synchronization routines. Therefore, to simulate other Master-Slave controllers, we assume that the average overhead O_M per request is a percentage of the time required to process only that request, that is if the expected processing time of a single request is μ_M and $O_M = 1(100\%$ overhead) is O_M, the expected time to process a new request will be $2\mu_M$.

To evaluate the article, we assume the following scenario. For common parameters such as main controller processing time, we assume a negative exponential distribution with an expected value of 20 μs. We simulate a network with 12 OpenFlow-compliant switches, each of which has a new flow arrival process (detected by the SDN controller) with an average value of 400 µs in all cases. Based on the presented results, the transmission delay in the OpenFlow channel is assumed to be uniformly distributed with a minimum of 4 microseconds and a maximum of 16 microseconds. Our proposed case evaluates two different probabilities of small packet loss L between the master and the datastore, which indicates the possibility of applying compatibility correction mechanisms.

The first probability is the packet loss probability of 0.00001 and the second probability is 0.05 (in real operational networks this value is not realistic, but is provided for illustration only). Finally, we test different overhead values to evaluate other Master-Slave controllers. We observe that among all the controllers evaluated, the delay of our Master-Slave controller has the closest performance to that of the unit controllers. We also observe that the probability of losing a packet does not have a large effect on the controller delay. However, it is shown that its effect on throughput is more noticeable. Finally, other master-slave approaches may operate with delay values close to the single controller if the overhead transmitted per replication is very low. However, obtaining small overhead values (eg less than 0.3) is a major technical challenge.

In this paper, we consider similar controllers, that is, single controllers, our proposal with packet loss probabilities of 0.00001 and 0.05, and other Master-Slave controllers with overhead values of 0.3, 0.5, and 0.7. We observe that under normal conditions, i.e. low probability of packet loss, our approach is the one that provides the closest output to that provided by a single controller. If the packet loss increases to very high values, the proposed Master-Slave output will be affected. However, this is not the case in real operational scenarios. On the other hand, the overhead provided in other Master-Slave controllers significantly reduces the output provided by them.

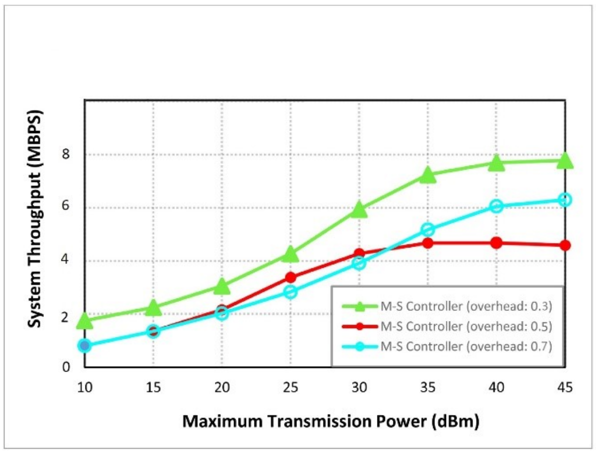

Figure 2: System total throughput vs Maximum transmission power.

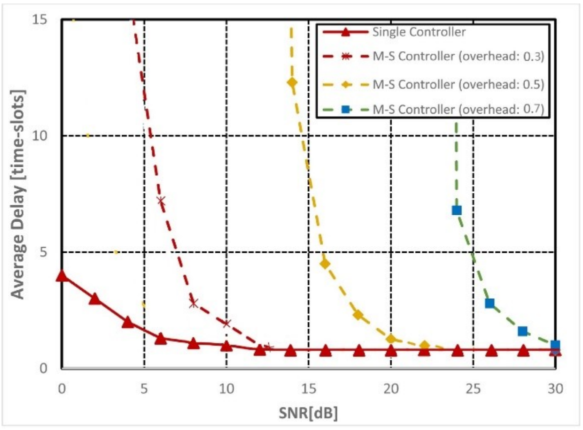

Figure 3: Average delay per signal to noise/interference ratio.

5 Conclusion

Answer to challenge 1: Our proposal does not have a single point of error and overcomes synchronization problems caused by delay differences and uncertainty in control processes. This is done by having a unique reference point and is complemented by additional tools to maintain consistency.

Answer to challenge 2: The proposed solution takes advantage of the potential simplification opportunities of the MasterSlave solution by providing consistency-aware routines, which are executed when needed, instead of being active throughout the entire controller operation. comes out

Answer to challenge 3: Our final solution is a hybrid method based on replication or replication of NIB updates, which ensures consistency and allows greater independence between masters and slaves. It is important to explain that the probability of some messages appearing twice in the design under our controller recovery routine is very low. This means that some rules may be rewritten, but at the same time, customers will not experience more downtime.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] O.A. Akanbi, A. Aljaedi, X. Zhou, and A.R. Alharbi, *Fast fail-over technique for distributed controller architecture in software-defined networks*, IEEE Access **7** (2019), 160718–160737.

[2] A.J. Gonzalez, G. Nencioni, B.E. Helvik, and A. Kamisinski, *A fault-tolerant and consistent SDN controller*, IEEE Glob. Commun. Conf. (GLOBECOM), IEEE, 2016, pp. 1–6.

[3] T. Hu, P. Yi, Z. Guo, J. Lan, and Y. Hu, *Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks*, Future Gener. Comput. Syst. **95** (2019), 681–693.

[4] T. Hu, P. Yi, J. Lan, Y. Hu, and P. Sun, *FTLink: Efficient and flexible link fault tolerance scheme for data plane in software-defined networking*, Future Gener. Comput. Syst. **111** (2020), 381–400.

[5] R. Kumar, M. Hasan, S. Padhy, K. Evchenko, L. Piramanayagam, S. Mohan, and R.B. Bobba, *End-to-end network delay guarantees for real-time systems using SDN*, IEEE Real-Time Syst. Symp. (RTSS), IEEE, 2017, pp. 231–242.

[6] K. Lee, M. Kim, H. Kim, H.S. Chwa, J. Lee, and I. Shin, *Fault-resilient real-time communication using software-defined networking*, IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS), IEEE, 2019, pp. 204–215.

[7] Q. Li, Y. Liu, Z. Zhu, H. Li, and Y. Jiang, *BOND: Flexible failure recovery in software defined networks*, Comput. Networks **149** (2019), 1–12.

[8] Y.D. Lin, H.Y. Teng, C.R. Hsu, C.C. Liao, and Y.C. Lai, *Fast failover and switchover for link failures and congestion in software defined networks*, IEEE Int. Conf. Commun. (ICC), IEEE, 2016, pp. 1–6.

[9] A. Malik, B. Aziz, M. Adda, and C.H. Ke, *Smart routing: Towards proactive fault handling of software-defined networks*, Comput. Networks **170** (2020), 107104.

[10] A. Mantas and F. Ramos, *Rama: Controller fault tolerance in software-defined networking made practical*, arXiv preprint arXiv:1902.01669.

[11] P.M. Mohan, T. Truong-Huu, and M. Gurusamy, *Fault tolerance in TCAM-limited software defined networks*, Comput. Networks **116** (2017), 47–62.

[12] L.F. Müller, R.R. Oliveira, M.C. Luizelli, L.P. Gaspary, and M.P. Barcellos, *Survivor: An enhanced controller placement strategy for improving SDN survivability*, IEEE Glob. Commun. Conf., IEEE, 2014, pp. 1909–1915.

[13] Y. Narimani, E. Zeinali, and A. Mirzaei, *QoS-aware resource allocation and fault tolerant operation in hybrid SDN using stochastic network calculus*, Phys. Commun. **53** (2022), 101709.

[14] S. Paris, G.S. Paschos, and J. Leguay, *Dynamic control for failure recovery and flow reconfiguration in SDN*, 12th Int. Conf. Design Reliable Commun. Networks (DRCN), IEEE, 2016, pp. 152–159.

[15] A.U. Rehman, R.L. Aguiar, and J.P. Barraca, *Fault-tolerance in the scope of software-defined networking (SDN)*, IEEE Access **7** (2019), 124474–124490.

[16] S. Song, H. Park, B.Y. Choi, T. Choi, and H. Zhu, *Control path management framework for enhancing software-defined network (SDN) reliability*, IEEE Trans. Network Service Manag. **14** (2017), no. 2, 302–316.