Fault Tolerant Traffic Engineering in Software-defined WAN*

Keyur Golani[1], Kunal Goswami[2], Kalgi Bhatt[3], Younghee Park[4]
Computer Engineering Department
San Jose State University
Email: {keyur.golani, kunal.goswami, kalgi.bhatt, and youghee.park}@sjsu.edu

***Abstract*— Software-defined networking in a wide area network (SD-WAN) allows intelligent control and management of networking, and efficient utilization of network resources through traffic engineering in real time for higher performance WANs. This paper proposes a fault-tolerant reactive routing system, called a smart routing system, for SD-WAN by investigating a variety of network features to be needed for monitoring in WAN in real time. The system keeps track of various network status data in real time to provide less packet loss and low network latency along with high availability and reliability in Software-defined WAN. We evaluate our system in real network provided by OpenLab at Juniper. Experimental results show that our approach successfully demonstrate resilience and efficiency by applying the programmability of SDN for WAN.**

Keywords—software-defined WAN; traffic engineering, OpenLab, network monitoring, reactive routing

I. INTRODUCTION (*HEADING 1*)

Software-defined networking in a wide area network (SD-WAN) functions to make a WAN more intelligible by decoupling the network hardware from the control plane and abstracting lower-level functionality. Enterprise-grade networks are moving toward higher performance WANs using low cost internet access. Gartner anticipates that 30% of enterprises will deploy SDN-WAN technology within a decade [17]. Its successful deployment and high quality of service (QoS) requires dynamic resource sharing through load balancing and resilient communication through multiple connection types using MPLS.

SDN provides many features that focus on improving the limitations of traditional networks. Using concepts associated with SDN such as programmability, single point of control, and seperation of control and data plane, the limitations of QoS in traditional networks can be improved. [12][13] present an approach extending SDN to WAN (Wide Area Network) using a dynamic routing algorithm. They have carried over a shortest path algorithm approach to SDN-WAN, calculating the optimal path from source to destination for a given current network status and flow rules. This leads to a reliable bandwidth and stabilizes QoS. Using this bandwidth and link failure, various performance analyses have been presented to support QoS in SDN-WAN.

This paper addresses a fault-tolerant reactive routing system called smart routing in SD-WAN. The system keeps monitoring network status through a local server (i.e. Redis) connected to network devices as well as real-time network information based on SDN. We design a cost function for smart routing based on network parameters and reliability parameters. The network parameters include network latency, the number of hops and the physical distance between two nodes while the reliabiltiy parameters are related to reliability of network links by considering various historical data. Since network links often fail, packets get lost while being sent from one endpoint to another endpoint in the network. It is very important to handle these failures in real time for high QoS. We modify a shortest path algorithm with the network and the reliability parameters in order to provide smart routing in the SD-WAN. The smart routing algorithm suggests the best path to carry out a communication for a given latency and reliability. Therefore, this algorithm incorporates the frequency of link failures and latency factors into the weight calculation and selects the path with the most negative weight as the best path for routing.

Two major contributions of this paper are providing a smart routing algorithm for SD-WANs and providing a prototype implementation. The smart routing algorithm is a unique approach which takes into account various network specific parameters and implements them in context with a modified shortest path algorithm. In addition, in spite of limitations of testing a new algorithm in a SD-WAN, we evaluate our proposed method by using OpenLab by Juniper Networks.

The rest of the paper is organized as follows: section II discusses the proposed method, followed by performance evaluation of the algorithm in section III. Discussion and related work is presented in section IV and lastly, section V discusses the conclusion.

II. THE PROPOSED METHOD

A. Overview

With SD-WAN, the same logical control can be obtained over a set of networks connecting different branches or offices or sub-networks together. With the rise in cloud infrastructure, enterprises delegate the task of deploying a scalable and flexible network to the cloud rather than setting it up using proprietary hardware. However, such wide scaled networks have wide scaled problems as well. The first and foremost one is of reliable communication, in this context, reliability does not only refer to a secure communication channel but it also refers to selecting a path which does not need retransmission of the packets. This section discusses our proposed framework for reliable and fault-tolerant data transmission in a WAN.

B. System Architecture

This section discusses the component details of the proposed framework: the network monitoring module, the event handler

978-1-5386-6950-1/18/$31.00 ©2018 IEEE

module, and the decision-making module in Figure 1. The modified routing algorithm needs a set of input parameters and an output identifying the best possible path. The inputs for this algorithm are formed with the help of the network monitoring module, and the event handler module. The network monitoring module collects statistics for the network and topology information, and the event handler module monitors the network for link failures. The network parameters and the reliability parameters computes the intermediate results for the algorithm by considering various sets of network feature sets in SD-WAN. We modify a shortest path algorithm with the weights per link depending on the various network statistics. With these inputs, the decision-making module then selects the best possible path for routing under various circustances in real time.

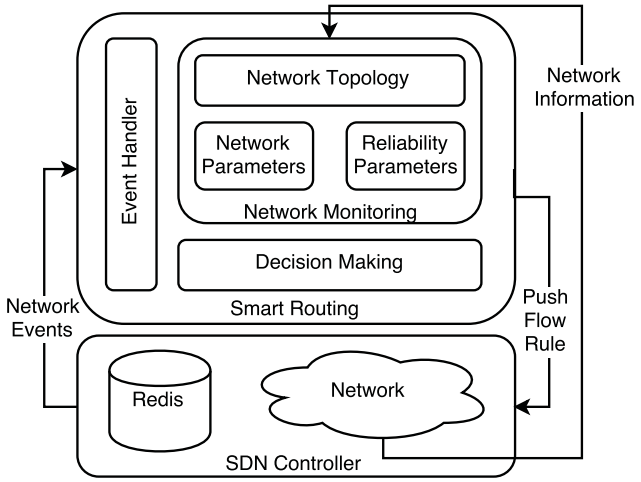

Fig. 1. Proposed System Architecture

1. *Network Typology Information:* The network monitoring module collects topology information and network information from the SDN controller to create a graph of the entire network. Since the algorithm determines the best path, it is necessary to represent the network with the help of a graph data structure to identify geographical physical distances. The default weight associated with each link is zero, as there is no network information available on setup.

2. *Event Handler:* The event handler module monitors the network for various link failures and recovery events happening across the network. The number of times a link fails in unit time can be considered as the failure frequency. With the help of failure frequency, a reliability factor can be associated with the particular link to determine whether or not it can be used for communication in the future. The reliability factor can be interpreted as an additional value associated with the link. Given a set of paths with the same overall weight, the path with the highest reliability should be selected for communication. The event handler module is connected to the Redis channel, which keeps track of all such events and acts as an event listener. The listener then calls respective APIs to accommodate the link failure and recovery events into the network graph generated by the network information module.

3. *Network Parameters:* Industry standards are used to set the various network features for choosing the best path. We define a set of network parameters related to network features. The network parameters include *link latency*, *link physical distance*, and the number of hops (i.e. *hop count*). Link latency gives an instantaneous performance prediction for the system. It is critical to consider latency when designing the system, as low latency is the key parameter in network QoS. Link physical distance is usually in direct proportion to link latency and so can usually be ignored, but we include it in order to handle real-world networks better. The belief here is that links at larger distances will have a higher probability of failing simply because they have more area where something could go wrong. The hop count simply represents the overhead that is able to be introduced at each hop. Here, the packets need to be routed by each device which could introduce some latency of its own. These three together constitute the set of network parameters that directly affect the delivery of the traffic.

4. *Reliability Parameters:* Reliability parameters, on the other hand, focus more on factors that will find a reliable link rather than the best immediate result. These parameters include *link failure*, *link failure frequency*, *link failure duration*, *past reliability record of the links*, *router input-output packets* (i.e. incoming and outgoing packets at a router), and *router input-output byte rate*. Link failures refers to the number of total failures a link has experienced since the beginning of the aggregation, i.e., the beginning of each day. For the same duration of time, link failure frequency will indicate the extent to which link performance fluctuates, which is an important factor in determining the reliability of a given link. Link down time from the beginning of the aggregation will inform how much time the link needs on average to recover from the failure, also contributing to the reliability factor of the link. And finally, a failsafe is needed since all of the parameters could overwhelm certain paths / links that have been performing well and reliably. To put a failsafe in place, a comparison of statistics for router input and output packets can give good insight into the system that is saturating a particular path and could be helpful in selecting another route for part of the traffic in such cases.

5. Decision-making: The decision-making module implements the smart routing algorithm proposed in this paper. The previous modules serve as the basis for computing the output for this algorithm. Network and reliability parameters form the foundation of the proposed algorithm. The decision is then made as to the best path for communication across the network, according to network latency and link distance as well as on the reliability of the link. It is essentially a modification of Dijkstra's shortest path algorithm, wherein we consider an additional set of weights for each link and update these weights periodically to make the correct decision at any given point of time.

The collective weight of link latency, physical distance, and hop count is drawn from the basis on which the link weights are updated. One important difference from the shortest path algorithm is that the weights are calculated multiple times to determine the optimal path at any given point in time. The algorithm is inspired by reinforcement learning with the help of interaction and feedback with a corresponding environment [18]. Initially, the links are each given equal weight, for example, if there are $\boldsymbol{n}$ links, each link is given a weight of one unit, as there is no information with which one can determine the parameters for an individual link. As time passes and the network functions normally, the weights are updated with the help of the following equation:

$$w = w + f(\lambda, d, h)$$

where, $\boldsymbol{\lambda}$ is the vector of the latency for each path, $\boldsymbol{d}$ is the vector of physical distances of the paths and $\boldsymbol{h}$ is the number of hops per path. Hence, $\boldsymbol{f(\lambda, d, h)}$ is the function that calculates the value by which the weights need to be updated. This function determines the mean and standard deviation of each one of these vectors and estimates how far each one of the entries within the vector varies from the mean. The more negative or positive the value obtained, the better the algorithm can estimate efficiency. The function $f(\lambda, d, h)$ is calculated using the following formula:

$$f(\lambda, d, h) = \left(\frac{\lambda - \mu_{(\lambda)}}{\sigma_{(\lambda)}}\right) + \left(\frac{d - \mu_{(d)}}{\sigma_{(d)}}\right) + \left(\frac{h - \mu_{(h)}}{\sigma_{(h)}}\right)$$

where, $\mu_{(\lambda)}$represents the mean of the vector λ, $\mu_{(d)}$represents the mean of the vector d, $\mu_{(h)}$represents the mean of the vector $\boldsymbol{h}$, $\sigma_{(\lambda)}$ represents the standard deviation of the vector λ, $\sigma_{(d)}$represents the standard deviation of the vector $\boldsymbol{d}$ and $\boldsymbol{\sigma_{(h)}}$ represents the standard deviation of vector $\boldsymbol{h}$. This procedure does not take into account link failures. It is certainly possible that with the help of the above equation, the algorithm would figure out the path with optimal or minimal latency. However, such an optimal path is not of much use if it fails too often. It is possible to build ***a pattern*** from the number of times the link has failed in the past and the duration of the failures. Both these values indicate the reliability of the link. If the link has failed too many times in the past, it is considered an unreliable link. Also, if the link has failed at repeated intervals, it will have a high link failure frequency indicating an unreliable link. The computer network selects the best path to communicate or transfer the information from source to destination. It is unacceptable for the link to fail during this transmission, no matter how minimal the latency.

The intelligence factor is therefore formed with the help of the reliability parameters of link failure frequency, duration of the failure, and the past reliability record if there is one. Each link is assigned a reliability attribute. Initially, all links are considered equally reliable, but the reliability factor is updated every time a link fails. This reliability update indicates how well the link performs over some unit of time, and if the link is performing well, the factor is updated accordingly. The update is a simple increment/decrement for link failure and recovery events. It is possible to include multiple variables for calculating the link reliability factor. Control passes over to the basic shortest path algorithm to determine the set of optimal paths for communication. The first choice would be the most negative weighted path from all the ones listed. Another factor considered is link usage: it is possible that a certain link with a very high reliability could end up carrying all network communication, leading it to become highly congested, which in turn would impact overall QoS in the SD-WAN. To avoid such a scenario, the network information module considers packets and bytes for each link, so that the second or third best path would be selected based on usage.

III. Evaluation

In this section, we evaluate our proposed system in SD-WAN provided by Juniper Networks' OpenLab to test the prototype in a real-world environment at Juniper Network Inc. [6]. Under the different real-world networks and scenarios, we evaluate our system in various angles, such as network latency, packet loss, and link failure rates. OpenLab [6] is an open network structure provided to educational institutions and other organizations for testing, competitions, and educational purposes. We used this network to evaluate our prototype in a real-world network environment to ensure the reliability and relevance of the approach.

A. Setup

The testing setup consisted of the network topology provided by the Juniper OpenLab [16] team along with the Northstar controller [14] and REST API [15] interface used to obtain network information as well as to configure the network. Juniper OpenLab is an environment provided to academic bodies and corporations for testing and debugging network-related systems with a real world network scenario. Northstar is the WAN SDN solution provided by Juniper Networks, designed to provide visibility and flexibility to large enterprise networks to control IP/MPLS flows. Northstar is the first traffic optimization WAN SDN controller in the industry. It automates traffic engineering paths across the network and provides flexible and programmable networks giving a customized network experience. The Northstar controller that we used was customized to not interfere with our experiments. The customizations mostly included stripping off the controller of all routing decision-making algorithms and making sure that the router would only accept the static path input from the REST API in order to decide the path for traffic to be routed. A GUI for the Northstar controller was also available to visually debug the network state for faster debugging and easier monitoring. However, this controller would not accept any manual configuration changes from the GUI.

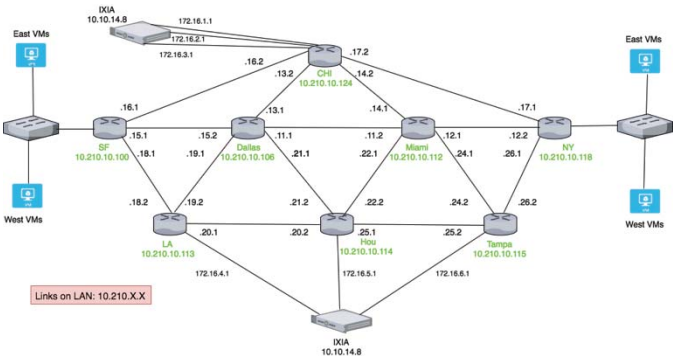

Fig. 2. Test Environment Network Topology

For network statistics and monitoring purposes, a Redis server was available which was useful in tracking network statistics, such as router ingress and outgress packets, and bytes. The Redis server also offered subscription to a notification channel using a publisher and subscriber model to track live link failure and recovery events. The statestics provided by the Redis server included network statistics including router incoming packets, outgoing packets, incoming bytes, outgoing bytes, link latencies over a 24 hours span, and physical distance between two router nodes.

The testing environment we used included a total of eight routers distributed around different states in the U.S. Routers at each node were connected to each other for failover connectivity simulating a real WAN network scenario. Two IXIA routers were connected at the vertical edges of the network to provide high volume vertical traffic adding to the testing traffic to simulate real world traffic flow in a network. For testing purposes, we are given access to both endpoints of the network, at NY edge and at SF edge router. This included multiple virtual machine accesses at the San Francisco end of the network attached to the SF router, and multiple virtual machine accesses at the New York end of the network attached to the NY router. The virtual machines contained raw Ubuntu 14.04 distribution that provided us a raw platform on which to set up our testing environment. The network situation we simulated over the provided infrastructure was one where one linked failed every 3 minutes, on average, and one link recovered from failure every 2 minutes, on average. Here, we configured our testbed for the first 10 lowest latency links to fail more than the other links. To generate various types of traffic simulating different cases including remote access, online gaming, and streaming, we used our own traffic generation scripts as well as Scapy and Ostinato tools as network traffic generators.

B. *Experimental Results*

The proposed system focuses on improving the choice of path over time by learning from past link failures and predicting better performing links. Hence, we expected the system to lower the number of packets lost, compared to a network without our proposed solution. We also expected the system to decrease packet loss even more over time by making trade-offs between low latency paths and reliabile paths. For this, we performed three experiments.

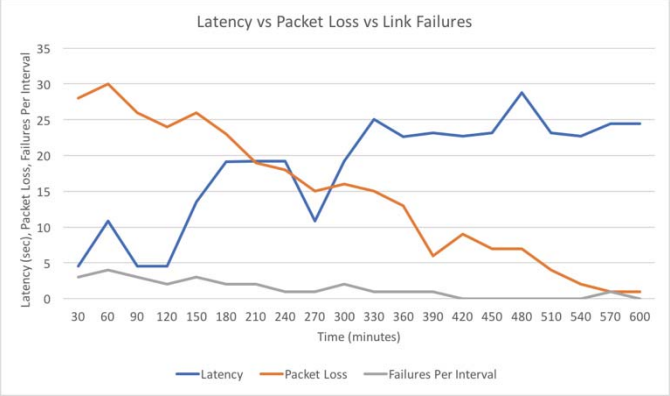

Fig. 3. Latency vs Packet Loss vs Link Failures

In Figure 3, we evaluated the overall effect of the proposed system related to network parameters first. We monitored the latency of the path chosen by the algorithm, the number of packets lost, and the number of link failures for a chosen route. Here, we expected the algorithm to initially choose low latency paths because reliability matrices were not yet built. We expected the algorithm to later gradually build reliability matrices and make trade-offs between reliability and latency. Then, the algorithm could choose routes with a little higher latencies that would still be able to avoid failed links, ultimately minimizing the number of packets lost.

As shown in Figure 3 initially when the algorithm was deployed, as expected, the paths chosen had the lowest latencies but the number of failures the chosen path experienced was high and hence the packet loss was also higher. Over time, reliability matrices were formed and the algorithm started choosing paths with slightly more latencies in order to avoid frequent link failures to save packet loss. Soon enough, the number of link failures for a chosen route approached zero for many consecutive intervals and hence the number of packets lost was also minimized to nearly zero.

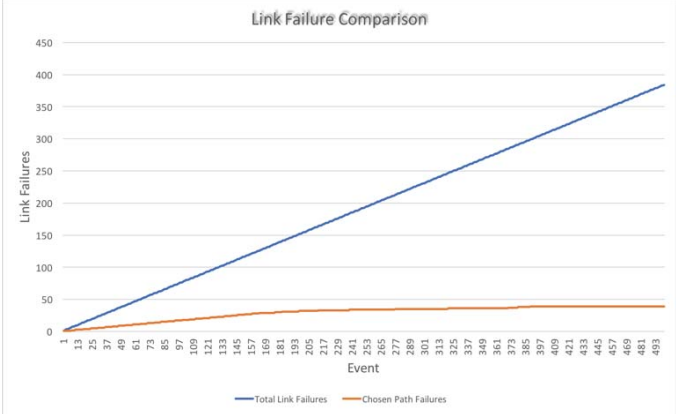

Fig. 4. Link Failure Comparison Results

In Figure 4, we measured the number of links failing on a chosen route as compared to the amount of links failing overall over the network. Here, we expected that the link failures over a chosen route would increase as reliability matrices were formed and then would become relatively constant. Figure 4 shows that the number of link failures in the chosen route increased over time. However, it started to get relatively constant very quickly, which suggests that there were minimal new failures over the chosen path. This result indicates long-term reliability and efficiency of the network.

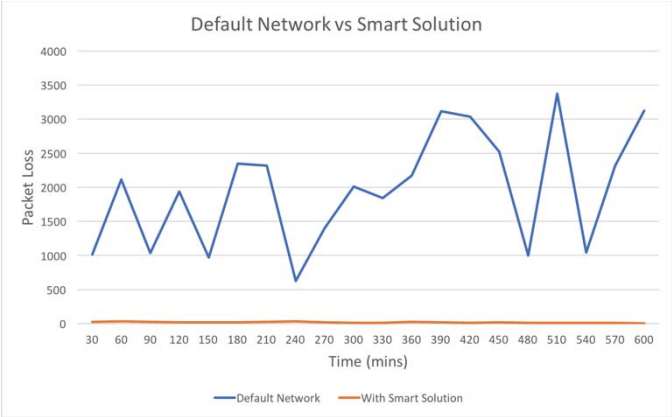

Fig. 5. Packet Loss - Default vs Smart

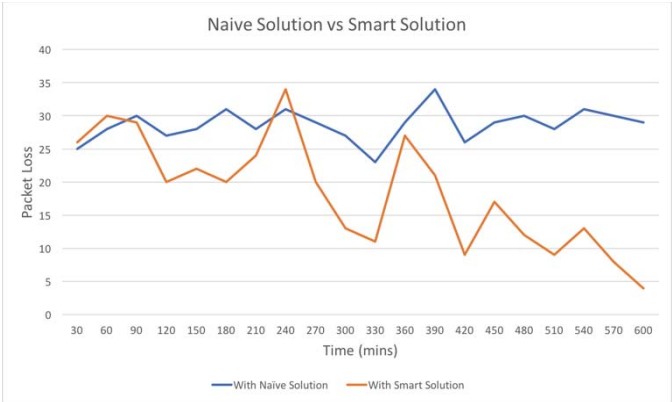

Fig. 6. Packet Loss - Naive vs Smart

In the experiments for Figure 5 and 6, we compared the overall number of packets lost in the network with and without our solution deployed. We devided the experiments into three scenarios: a default solution, a naïve solution, and a smart routing. The default solution in a network is the original network in OpenLab that Juniper Network Inc. had set up without our entire smart routing solution. The naïve solution only consider the network parameters without the reliability parameters. The smart routing solution is our poposed entire system. Therefore, Figure 5 and Figure 6 showed the results of the combination of the three scanarios.

Here, we expected that compared to the default network, the number of packets lost would be drastically minimized because of the presence of rerouting and failure handling mechanisms in the network. Compared to any naïve solution deployment, we expected our algorithm to perform better over time. Figure 5 shows that with default network, the number of packets lost was reduced to the scale of 0.01, which met our expectations. Figure 6 shows that when deployed against a naïve solution, our solution starts with almost the same number of packets lost. However, our proposed smart routing algorithm learns from link failure history and avoids failed links over time, resulting in fewer overall packet losses.

IV. Discussion and Related Work

A. Discussion

Software Defined Networking is a rapidly emerging paradigm in local networks and is also gaining much popularity in large corporate networks. The concept of SDN separates the data plane and the control plane on networks, which increases network control in programming and and design flexibility. However, SDN supplementing WANs is a relatively new concept. Because a WAN testing environment is more difficult to set up and maintain, this concept remains relatively unexplored. WAN networks are large, convoluted networks that usually suffer much from delays, jitters, and other QoS degradents. However, they also accommodate several alternative routes, connecting each pair of nodes that could be utilized to advantage. SDN can provide a WAN network with programming flexibility and configurability of widely divergent network configurations, which could never be achieved manually. In this proposal, we use this advantage of SD-WAN (Software Defined Wide Area Network) to configure the routes in WAN automatically in such a way that the low performing corners of the network can be avoided altogether and by doing so, the effects of link failures on QoS can be minimized.

Two major contributions from our side are the smart algorithm inspired from reinforcement learning to determine the best path to choose for routing and testing in a real world test environment provided by Juniper's OpenLab [16]. The smart algorithm takes into consideration latency, hop count, and distance as well as device and link reliability parameters including link failures, failure frequency, and link down time. The real world test environment is a WAN implementation across the United States with a live Northstar WAN SDN controller [14].

Our proposed system predicts the best routes to take and makes sure that reliable routes are chosen from all available routes. However, more work could be done in making the algorithm even smarter to ensure that no route in effect ever fails. In the future, we intend to extend this approach and integrate algorithms to monitor the health of network devices, their behavior, and more importantly, their behavior patterns along with predictive analysis to get pre-notified of any failure or roadblock in the network. This could help in making changes to chosen paths before a link fails and avoid even a small amount of packet loss in brief time between link failure and new route choice. We also intend to propose an actual reinforcement learning agent developed from the proposed algorithm in this paper. Integrated with a predictive analysis algorithm, this could result in more intuitive and effective decisions in terms of avoiding unreliable areas of the network.

B. Related Work

Traffic engineering and management are an important concept to consider in SDN [7, 8, 9, 10, 11]. S. Agarwal and et. al. reduced packet loss and delay by utilizing a centralized SDN controller and by managing network traffic under changing conditions [11]. Slavica and Neeli mainly focused on collecting network information, making smart management decisions based on that information, and providing a centralized system, all of which helps in minimizing the degradation of network traffic [10]. Frederic and et. al focused on implementing energy-

aware routing using an SDN controller using link capacity and flow rules. This energy-aware routing saves energy by putting links that are not currently being used into sleep mode. They used SDN to collect network statistics to calculate the routing using QoS to save on energy [8]. Andreas and et. al. have proposed an algorithm to handle low-level details in a complex network [7]. It is an algorithmic policy that handles SDN controller and simplifies it by providing a centralized algorithm that can decide the behavior of the network.

Addressing availability issues of SDN and developing a highly available application using SDN is very important. Aditya and Arvind [5] stated that improving the scalability and performance of SDN will yield high availability even though there are many failures in the network. They designed a fault-tolerant SDN fabric application that increases the availability of SDN by recalculating the SDN architecture and avoiding network failures.

SDN provides programmability, availability, and performance to networks, but at the same time there are several challenges to be overcome. G. Nencioni and et. al. discusses an approach to compare the features provided by SDN to traditional networks [1]. Following this, Park first identified issues related to high availability in SDN and then proposes algorithms such as the Cluster Virtualization algorithm, Cluster Information Consistency Algorithm, Detection Algorithm (running on the controller) and Detection Algorithm (running on OpenFlow switch). Wenbo and et al. [4] proposed a new architecture of BGP called OFBGP, which is an application for a SDN controller extending the availability and scalability properties of the SDN controller [4]. They also implemented a prototype of their architecture and the results obtained by experimenting on this prototype provide a highly scalable and available architecture for BGP by extending the high availability and scalability of the SDN controller.

Seyhan and Murat discuss an approach showing how inter-domain routing takes place using SDN, keeping availability of resources of SDN in mind [3]. In the paper, the authors proposed an autonomous decision-making system that can make decisions and choose the path for the flow among multiple domains using SDN. In a similar way, Subhasis and Kalapriya focused on managing the flow table efficiently in SDN switches, extending the availability of SDN [6].

SDN provides many features that focus on improving the limitations of traditional networks. Using SDN concepts such as programmability, single point of control, and separation of control and data plane, limitations of QoS in traditional networks can be improved. Recent research presented an approach extending SDN in WAN (Wide Area Network) using a dynamic routing algorithm [12, 13]. They have used a similar approach to using the shortest path algorithm in SDN-WAN calculating the optimal path from a source to destination considering the current network status and flow rules. This will lead to a reliable bandwidth, stability, and QoS. Using this bandwidth and link failure, various performance analyses have been presented for supporting QoS in SDN-WAN.

REFERENCES

[1] G. Nencioni, B. E. Helvik, A. J. Gonzalez, P. E. Heegaard and A. Kamisinski, "Availability Modelling of Software-Defined Backbone Networks," *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, Toulouse, 2016, pp. 105-112.

[2] H. Park, S. Song, B. Y. Choi and T. Choi, "Toward control path high availability for software-defined networks," *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, Kansas City, MO, 2015, pp. 165-172.

[3] S. Civanlar, E. Lokman, B. Kaytaz, A. Ulaş and A. M. Tekalp, "Distributed flow management for SDN domains," *2015 23nd Signal Processing and Communications Applications Conference (SIU)*, Malatya, 2015, pp. 2609-2612.

[4] Aditya W. Duan *et al*., "OFBGP: A Scalable, Highly Available BGP Architecture for SDN," *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, Philadelphia, PA, 2014, pp. 557-562.

[5] Aditya Akella and Arvind Krishnamurthy, "A Highly Available Software Defined Fabric," *2014 Proceedings of the 13th ACM Workshop on Hot Topics in Networks* (HotNets-XIII), *ACM*, New York, NY, 2014, pp. 7-21.

[6] S. Banerjee and K. Kannan, "Tag-In-Tag: Efficient flow table management in SDN switches," *10th International Conference on Network and Service Management (CNSM) and Workshop*, Rio de Janeiro, 2014, pp. 109-117.

[7] Andreas Voellmy, Junchang Wang, Y Richard Yang, Bryan Ford, and Paul Hudak, "Maple: simplifying SDN programming using algorithmic policies," *2013 Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (SIGCOMM '13), ACM, New York, NY, 2013, pp. 87-98.

[8] F. Giroire, J. Moulierac and T. K. Phan, "Optimizing rule placement in software-defined networks for energy-aware routing," *2014 IEEE Global Communications Conference*, Austin, TX, 2014, pp. 2523-2529.

[9] R. Wang, Z. Jiang, S. Gao, W. Yang, Y. Xia and M. Zhu, "Energy-aware routing algorithms in Software-Defined Networks," *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, Sydney, NSW, 2014, pp. 1-6.

[10] S. Tomovic, N. Prasad and I. Radusinovic, "SDN control framework for QoS provisioning," *2014 22nd Telecommunications Forum Telfor (TELFOR)*, Belgrade, 2014, pp. 111-114.

[11] S. Agarwal, M. Kodialam and T. V. Lakshman, "Traffic engineering in software defined networks," *2013 Proceedings IEEE INFOCOM*, Turin, 2013, pp. 2211-2219.

[12] A. M. Al-Sadi, A. Al-Sherbaz, J. Xue and S. Turner, "Routing algorithm optimization for software defined network WAN," *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*, Baghdad, 2016, pp. 1-6.

[13] S. Tomovic, I. Radusinovic and N. Prasad, "Performance comparison of QoS routing algorithms applicable to large-scale SDN networks," *IEEE EUROCON 2015 - International Conference on Computer as a Tool (EUROCON)*, Salamanca, 2015, pp. 1-6.

[14] "NorthStar WAN SDN Network Controller - Juniper Networks," NorthStar WAN SDN Network Controller - Juniper Networks. [Online]. Available: https://www.juniper.net/us/en/products-services/sdn/northstar-network-controller/. [Accessed: 14-Apr-2017].

[15] "Juniper Northstar API Documentation," Juniper. [Online]. Available: http://www.juniper.net/techpubs/en_US/northstar2.1.0/information-products/api-ref/api-ref.html. [Accessed: 14-Apr-2017].

[16] "OpenLab: Junos Center for Network Innovation - Juniper Networks", Juniper.net, 2017. [Online]. Available: https://www.juniper.net/us/en/company/openlab/. [Accessed: 15- Apr-2017.

[17] Gartner, "Predicting SD-WAN Adoption", December 15, 2015.

[18] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. Cambridge, MA: MIT press, 1998, vol. 1, no. 1.