Finding Most Reliable Paths For Software Defined Networks

Ali Malik, Benjamin Aziz and Mohamed Bader-El-Den
School of Computing, University of Portsmouth, Portsmouth PO1 3HE, UK
Email: {ali.al-bdairi, benjamin.aziz, mohamed.bader}@port.ac.uk

***Abstract*—In this paper, we introduce a new approach that computes the shortest-reliable end-to-end paths for centrally controlled networks like software-defined networks (SDNs). The proposed method aims to find the correlation between the routing mechanism and reliability with the purpose of decreasing the required time of backup path installation through reducing the number of required rules at the moment of failure towards guarantee the fast restoration of the affected path, hence leading to the reduction of the overhead on SDN network controller and the probability of the loss of packets. We also investigate the correlation between the network topology and its reliability and demonstrate the benefits from this relation through experiments using well known SDN network simulation tools.**

***Keywords*—Network Topology; Routing; Reliability Configuration; Software Defined Networks.**

I. Introduction

Distributed control systems in networking devices along with a set of defined protocols constitute a fundamental technology that have been adopted to send and receive data via networks around the world in recent years . Since the control system (known as the *control plane*) and the data forwarding unit (known as the *data plane*) are tightly coupled in appliances, this has led to rigidity in the classical IP network systems where each networking device can be seen as a closed box and the update operation can be done only by the manufacturer company [1]. Such networking devices exchange data in a variety of networks including wireless sensor networks, the Internet-of-Things, Cloud networks and so on. Currently, there are about 9 billion devices connected to the Internet and this number is expected to more than double by 2020. Traditional IP network infrastructures have no ability to support such a huge number of devices, which may cause Internet ossification [2].

The rigidity of the traditional network architecture has an impact on increasing the difficulty of handling the transferred data. According to [3], 62% of network downtime is caused by human errors while 80% of IT expenditure is on maintenance and operations, therefore, some attention must be given to find a more flexible solutions that could meet the future Internet and networking requirements. To surmount the inflexibility in the conventional network architecture, the emergence of Software Defined Networks (SDNs) has facilitated the separation between the control and data planes giving more ability to create a *programmable* network that can be managed like computing device. Recently, SDNs have gained the attention of both academia and industry, and this is due to the bunch of advantages that SDNs bring with them against the traditional networking system. For example, SDNs have been recently adopted by well known companies such as Google, Facebook and Microsoft. SDNs also constitute the underlying mechanisms for networking new generations of virtual containers, such as Dockers. Currently, the two main issues that stand out are reliability and fault tolerance, and they are widely considered as serious issues facing the SDN research community [4]. The main goal of this paper is to highlight the importance of the path selection operation and its significant role at the process of failure recovery through providing a new algorithm that achieves the reliable shortest path between any two given nodes. We have evaluated our new proposed algorithm through simulation experiments and the reported outcomes revealed an encouraging results.

The rest of this paper is organised as follows: In Section II, various fault tolerance techniques of SDNs are presented and discussed. Then we discuss some performance evaluations for existing SDN environments in Section III. In Section IV, types of data transmission paths with criteria of path selection are presented. Section V gives an expanded view on the SDN network's topology. Our method and algorithm are presented in Section VI and VII respectively. Finally, the summary of this paper and future work are provided in Section VIII.

II. Related Work

The topic of SDN failure management has been already studied and we will critically discuss the relevant literature in this section. Due to the fact that a SDN architecture decouples the control plane and data plane, thus, failures could affect the both planes (e.g. controller, nodes and links), whilst our work concerns with the data plane issues through tackling the problem of single link failure. So far, the OpenFlow protocol [5] is considered as the most commonly used protocol for enabling the network controller to dictate the data plane elements that lie on its domain as well as governs the communication interface between the two separated planes. As a result, the data plane lacks of the decision-making feature due to the shifting of the control unit to the controller entity. Therefore, the controller intervention is required at every single network event (e.g. failure events).

The mechanisms of data plane (i.e. links and nodes) failure recovery can be classified into *protection* and *restoration* techniques [6]. In protection techniques, the controller computes all possible alternative paths and installs the forwarding rules

for those backup paths in the forwarding elements before the failure event occurs. Hence, this technique will proactively mask the failure. On the other hand, the restoration mechanism behaves re-actively by telling the controller to install the proper rules for an alternative route at the moment of failure. In [7], the authors describe how it is hard to achieve a data plane restoration within $\leq$ 50 ms for large SDN networks, while it is achievable with a protection mechanism. According to [6], the restoration time could be in the range of 200–300 ms. In the same context, the authors in [8] show that the maximum time of data plane protection can be achieved within less than 64 ms. thus, the data plane protection mechanism is more preferable especially for large-scale SDNs due to the time issue that required to perform the restoration.

The Adaptive Multi-Path Computation Framework (ADMPCF) [9] for large scale OpenFLow network systems has been developed as a traffic engineering tool for SDNs, which capable to hold two or more disjoint paths to be utilised at the moment of some network events (e.g. link failure). In addition, CORONET [10] has been proposed as an SDN fault tolerance system to solve the issue of multiple links failure through finding multiple disjoint paths. As a result, the preserved paths in both of the aforementioned works can be employed as a backup in case of link/node failure or when the cost function does not meet the standard definition, hence the ADMPCF and CORONET behave reactively to mask the data plane failures.

Our work is relatively close to [11], in which the problem of minimising the number of operations for the backup path is introduced. Instead of dealing with such a problem at the moment of failure, our propose method deals with the problem from an early stage, in which we have considered the operation of path selection as an essential step that must be thoroughly investigated to select the route carefully rather than waiting till the moment of failure. Hence we are aiming to guarantee the feasibility of the minimal number of flow entries that needed to mask the failure, which is currently unexplored.

Both protection and restoration techniques have pros and cons from the perspective of network performance and the Quality of Service (QoS). The main issue with the proactive methods is the memory space exhaustion, since the network controller manipulates the forwarding rules of the data plane nodes that stored as a TCAM (the Ternary Content Addressable Memory) entries, while the TCAM space is limited and expensive [6]. Rather than this, the installation of many attributes will affect the process of match and action of the forwarding elements in data plane. Also there is no guarantee that the preplanned paths are failure-free, in other words, the backup paths could fail earlier than the primary ones. On the other hand, the main disadvantage of the reactive methods is the time issue because the controller needs to receive an update about the topology after each failure event through exchanging several control messages between the data and control planes. Afterwards, the controller will recalculate a new valid path by removing and adding some new rules in order to continue conveying data successfully. Thus, reactive schemes might not be the ideal solution for a large-scale SDNs, so that this paper proposes a new method for mitigating the time issue of the restoration mechanisms through path selection as a technique.

III. Preliminary Evaluation

As mentioned earlier, the SDN controller will be responsible for the network intelligence and states through relying on the global view of the network topology. There are different types of messages that used by the controller to configure the data plane, which is usually take place via the OpenFlow protocol. When a new packet arrives to an OpenFlow switch, the switch will first check the packet header against all the preserved rules. If there is a match, then the switch will execute the matched rule action, otherwise the network controller will be asked on how to deal with the incoming packet via receiving a *packet-in* request from the particular switch. Thereafter, the controller will process the switch's request and respond by installing the proper rules through the *flow-mod* message as shown in Figure 1.

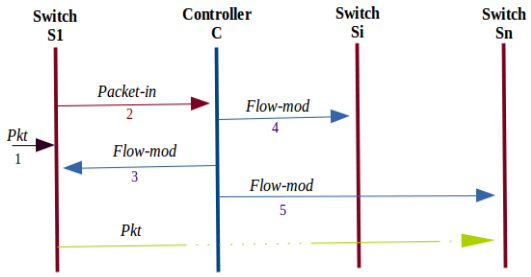

Fig. 1: *Steps of adding rules in an SDN*

In order to investigate whether the controller needs more time to set up long paths compared to shorter ones, we performed the following test as a preliminary evaluation.

- Emulation Testing Environment

The Mininet emulator [12] for a realistic virtual network creation has been utilised to perform this test. Three different linear topologies T1, T2 and T3 with 1, 10 and 50 switches were generated respectively and used in this experiment. Each topology was controlled by a one local POX controller [13], we then added two hosts that specified to generate a network packets, so we can measure the ping time along the full length of paths. It is worth noting here that the controller could be placed remotely, and also it may not connected directly to the switches that located on its domain [14]. After running the ping test, we have noticed that the time consumption of the first ping is much higher than the consecutive pings; which have an approximately equal time and that is because the flow entry rules were previously added to the particular node(s) and the packet immediately passes without need to ask the controller, hence only the results of first and second ping are reported. In addition, we noticed that the first ping time increases whenever the path length increased. The tests have been repeated up to 50 times to guarantee the accuracy, for T1 the average time of the first ping test was 3.297 ms and the second ping was

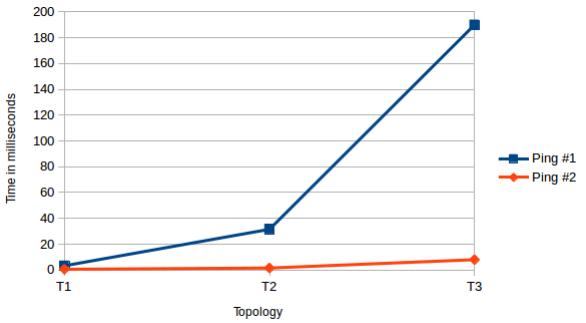

Fig. 2: *Ping test scenario*

0.606 ms, whereas for both T2 and T3, the first average ping time was 31.536 ms and 189.92 ms respectively, however the second ping time were 1.558 ms and 7.299 ms. Figure 2 shows that the time of path installation increased when the path length is long and decreased when the path is short. Let us assume the total time that required to install the path (P) from the source (S) to the destination (D) is (T), and also assume that (t) is the required time to install a single flow entry in an individual node (i), which belongs to the path P. So, T can be computed by the following equation:

$$T = \sum_{i=S}^{D} t(i) \tag{1}$$

According to (1), T is directly proportional to the length of P ($T \propto P$) and therefore, the path selection is a critical subject due to its correlation with a various network events such as link failure. The next section specified to introduce the criteria of path selection in networking systems.

IV. Path Selection Criteria

In networking systems, data can be disseminated (from source to destination) either through a *single* or *multi* path(s). In single path, data should be routed from origin to destination through a unique path, which has to meet some of predefined constraints. While the other solution for traffic distribution is by routing the traffic through a number of existing paths between the origin and destination.

According to the literature, for each of these routing strategies there are some different methods with common principles such as using the well-known shortest path algorithms like Dijkstra, Bellman-Ford, etc. The major routing schemes are usually focusing on how to select the most optimal path (single path) in order to disseminate the data packets [15], while various of single-path algorithms have been reported and classified based on their QoS metrics in [16].

On the other hand, the schemes of routing in which multi-paths (good but not optimal) should be selected to distribute the data packets rather than using only one optimal path have been widely studied in the literature (e.g. [17] [18]). Authors in [16] have classified the QoS based routing problems into three categories: Multiple constrained path, Multiple constrained optimal path and restricted shortest path. For all of the proposed algorithms there is a set of common metrics such as bandwidth, delay, jitter, packet loss and hop count. The concept of disjoint path, when $Path_1 \cap Path_2 = \emptyset$, can be utilised in both data transmission scenarios, for instance it can be employed as a backup for the single path methods or a primary/backup for the multi ones. Disjoint paths have a significant benefit over many aspects and it always more preferable to be used in the context of enhancing the network performance like link/node failure, loud balance improvement and for better network resource utilisation. The problem of finding network disjoint link/node has gained much attention in the literature (e.g. BGR [19] and KSP [20]) due to its theoretical and practical importance in many applications such as communication survivability and routing reliability [21]. In general, path selection process associated with cost, which is indicating the price of each path and it can be measured through a cost function, where the idea behind the cost function is that the punishment probability of assigning links in the network grows with the incompatibility ratio of the link against the cost function form. Later, we will answer the question of whether the disjoint paths are convenient to the nature of SDN architecture in terms of failure recovery.

V. Topological View of SDN

SDN is a centralised networking system, in which the controller (the brain of network) has to calculate the forwarding tables for all nodes that belongs to its domain. As mentioned before, the nodes will forward the incoming packets across the links based on the installed rule that can be added either proactively or re-actively. Paths usually consist of a set of successive edges that led to a particular destination where the number of edges in each path determines its length. In Figure 3, paths with various lengths have been created in order to clarify the relationship between the inserted rules and the number of hops in those paths.

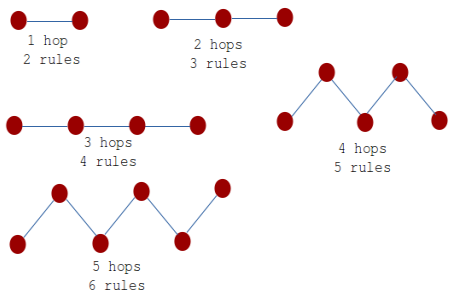

Fig. 3: *The relation between rules and hops*

It can be noticed that the number of inserted rules is always greater than the number of hops by 1. If we assume that N is the number of hops in a network Path P, so that the controller will need to add $N+1$ rules in order to setting up the P. This can be formulated as follows:

Let: $N \equiv number\ of\ hops$

$$N' \equiv N + 1 \quad number\ of\ rules \tag{2}$$

Thus, path selection is important especially in case of failure events and could plays a significant role to reduce the recovery time and mitigate the overhead of the network controller. To the best of our knowledge, none of the previous work on SDN failure recovery has considered the path selection as a technique to reduce the recovery time, hence this work can be seen as a first attempt in this context and will provide as such a cutting-edge contribution to SDN research.

Since reliability is an essential criterion for the network communication robustness and it is usually defined as the probability of connection continuity between the source node and destination [15], or the capability of delivering the services to users in a presence of network element(s) failure. Although the reliability has been defined and studied before, we will give it a new scope in the following section.

VI. Shortest Reliable Path

Our goal is to compute the most reliable route between any two given nodes in a network graph. In general, every simple graph $G = (V, E)$ is usually consisting of a set of vertices V and edges E, which connect the vertices to one another.

$$E \subset V \times V \tag{3}$$

According to (3), each edge $e \in E$ is a subset of $V \times V = \{(u, v) : u \in V,\ v \in V\}$, since a path is a sequence of consecutive edges in a graph G that means each edge e connects between every two adjacent nodes in G. In SDN, the total number of inserted rules into any path is greater than the number of hops in that path (2). Whilst the number of added rules can be reduced whenever there is a one (or more) shared link(s) between the primary and backup path. It is worth noting here that we are dealing with paths that have an equal number of hops, and later we will show the trade off between the shortest and reliable (non-shortest) paths. Figure 4 and 5 illustrates the one and two shared link benefits in decreasing the number of required rules.

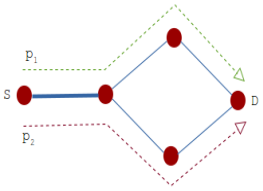

Fig. 4: *One shared link paths*

Let: $PATH$ be the set of all path names in G.
$No_{hops} : PATH \rightarrow N^+$
$No_{rules} : PATH \rightarrow N^+$
For P_1, P_2 in Figure 4:

$$No_{hops}(P_1) = No_{hops}(P_2)$$
$$No_{rules}(P_1) = No_{rules}(P_2)$$

It can be observed that there are two paths P_1 and P_2 in Figure 4, each has the same number of hops and there is a one shared link marked with a thick line between them. Assume that P_1 is the primary path and P_2 is the backup one (vice versa can be assumed). At the moment of link failure (except the shared edge in this case) of the primary one, the flow entry of node S will not require any change and therefore the controller will need to push 3 rules rather than 4, this is due to the common edge existence. While in figure 5, there are three paths P_1, P_2 and P_3 from the source node A to the termination node Z. Each path has the same number of hops and required the same number of rules to be installed if its assigned as a primary. For P_1, P_2 & P_3 in Figure 5:

$$No_{hops}(P_1) = No_{hops}(P_2) = No_{hops}(P_3)$$
$$No_{rules}(P_1) = No_{rules}(P_2) = No_{rules}(P_3)$$

Each path involves two types of edges: shared edges and unshared ones. There are two shared edges between P_1 and P_2, while the P_3 comprises a set of completely unshared edges. If P_3 will be selected as a primary path, then the controller will need to push a five new flow entries to mask the error at the failure moments of any edge in P_3. however if either P_1 or P_2 will be setted as a primary one, then any link failure (except the shared ones) will cost the controller only three rules rather than five, since the flow entries of node A and B will remain unchanged, and this is because of the two common edges between the P_1 and P_2. The number of unchanged rules increased proportionally with the number of shared edges, for example if the path has K shared edges then the saved rules should be K as well (with the assumption that the shared link is failure-free). Therefore, we could define the reliability as a link between any two source/destination nodes with more shared edges, so this definition is different from those used previously. To this end, we introduce the following formula regarding the relation between the rules insertion and shared edges (we have tested some other different types of topologies, but due to the limit of paper size we have not included them).

$$(S_r \propto S_e) \tag{4}$$

Where S_r represents the number of saved rules and S_e represents the number of shared edges. In the next section, an algorithmic method to pick the most reliable path in a SDN environments will be introduced.

VII. Reliable Path Selection

Currently, our developed algorithm is dedicated to minimising the time that's required to mask a single link failure only. Our solution is not concerned with how the SDN controller will detect the link failure or measures the time that's required for such a process, our solution focuses on how to select a primary path that could be utilised even at the moment of failure. In other words, we define the reliability as the ability to be relied on a solution that could be derived other solution(s) of it with less cost.

On the basis of the observation from (1), in which the time T increased as long as P length increased, we can assume

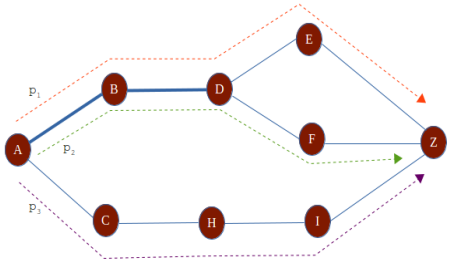

Fig. 5: *Two shared link paths*

that the disjoint path is not always the perfect choice as a backup because it will require a totally new flow entries for all nodes that form a path P. Hence, the recovery time could be increased depending on the length of P.

Furthermore, the shared edges between two paths demonstrate the ability to reduce the required rules (4) and for this purpose, we propose to select the path that has a maximum number of shared edges as a technique to reduce the T at the moment of failure occurrence, in other words, S_e can be defined as a concave metric while selecting the route. Additionally, the maximum shared edges is important not only to reduce T, but also to give the controller more options to reach the data plane elements in case of those elements miss the direct connection with the network controller (see [14]).

A. The Reliable Path Finder (RPF) Algorithm

The concept is simple in which the set of all paths from source to destination, which have been captured by the NetworkX [22] tool, need to be evaluated for the purpose of obtaining the most reliable path that has the highest number of shared edges. Mathematically, we can express this in terms of the following set relations.

Let *Edges*: $V \times V$ be an edge connecting two vertexes in the network graph. $\mathcal{P}_{edges} : PATH \rightarrow \wp(Edges)$ then defines for any path name taken from the set of all possibly paths, $PATH$, in terms of the set of edges that constitute that path. We assume that for any two nodes (vertexes), then $\mathcal{P}_{Set} = \{P_1, P_2, ..., P_n\}$ represents the subset of $PATH$ that contains all the possible paths linking those two nodes.

We define the intersection between any two paths as follows: $S_i = \mathcal{P}_{edges}(P_i) \cap \mathcal{P}_{edges}(P_{i+1})$. We require for our algorithm that $S_i \neq \emptyset$. We express the set of all possible intersections between any two paths as $\mathcal{S}_{Set} = \{S_1, S_2, ..., S_n\}$. We are interested in the intersection with the maximum number of shared edges between two paths. We capture this in terms of the following operator: $Max(S_{Set}) = S_j, where\ \forall i.\ i \in \{1, \ldots, n\}, i \neq j, S_i \in S_{Set} : |S_j| \geq |S_i|$, which returns the intersection with the highest number of edges.

RPF is an algorithm that receives the network topology, which is represented as a graph G, as well as the source and destination nodes. Since the SDN controller has a global view on the network's topology state, therefore the data plane topology can be utilised by the control plane applications for various aims. The pseudo code for our algorithm is summarised in Algorithm 1. The main functions in the algorithm are presented in line (1-6), the algorithm holds all possible shortest paths between any two given nodes in a graph G and forward them straight away to the function defined in line (1), which transforms each path into a set of edges. Finally, the RPF function (line 4) will be invoked to return the final reliable path that contains the highest number of shared edges.

Algorithm 1: RPF

input : Network *Topology*, *Source* and *Destination* nodes
output: Reliable path with maximum shared edges

```
1  Function Pairwise (P)
2      divide the path into pairs of edges
3      return Set of edges
4  Function RPF (Set_P)
5      capture the path who has a maximum shared edges
6      return Reliable path
7  for P in P_Set do
8      Set_P ← Pairwise(P);
9  end
10 RPF (Set_P)
```

B. RPF Performance Evaluation

This section shows the RPF impact on the performance of SDN in terms of link failure. To do so, the RPF has been implemented as an additional component, which is a real python module, on top of the POX controller. Actually, this module fetches the network graph (topology) via the *openflow.discovery module* that can be used to send LLDP messages out of the openflow switches in order to figure out the current network topology. After detecting the topology, the network graph G parameter will be updated and continue the subsequent steps as mentioned in the RPF algorithm. In order to evaluate the efficiency of RPF, different testing scenarios have been applied on various type of paths. We assume that both of *hard* and *idle* timeouts are infinity for each installed flow entry on our OpenFlow switches with a view to highlighting the concept of our method. Figure 6 summarises the results of ping time test for three different scenarios of our two empirical topologies as follows:

$case_1$: shows the average ping time of both primary and backup paths for the topology designed in Figure 4, in which the scenario of one shared link has been illustrated.

$case_2$: shows the average ping time of both primary and backup paths for the topology designed in Figure 5, which demonstrates the two shared edges. In this case the primary path is {A, B, D, E, Z}, while the backup {A, B, D, F, Z}.

$case_3$: shows the average ping time of both primary and backup path for the topology designed in Figure 5. In this case the primary path is {A, C, H, I, Z} and the time is approximately equal when the backup is {A, B, D, E, Z} and the same can be said when the path is {A, B, D, F, Z}.

Based on Figure 6, when we compare the difference of ping time between the primary path and the backup one, we observe that the shared edges has an impact on reducing the ping time, which means the operation of recovery from failure is sped up.

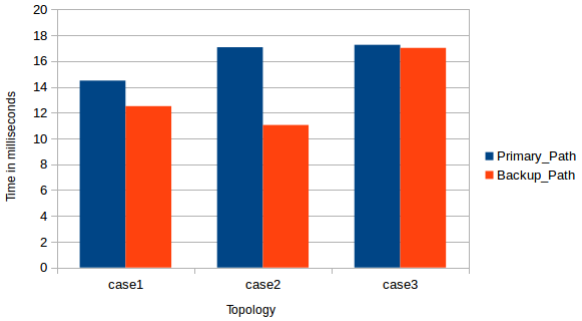

Fig. 6: *RPF evaluation*

Although the two of primary and backup paths in our scenarios have the same length, the ping time T of the backup path is inversely proportional to the number of shared links in the primary one, so we can formulate this as follows.

$$T = K/Number_{S_e} \tag{5}$$

Where K is the constant of proportionality. The main limitation of RPF is the inequality in length between the shortest path and the desired or preferable path. In such a case, the trade-off between the shortest path and the fast route recovery can be viewed as a requirement from the network operator. While, the reliability criteria has obviously some costs.

VIII. Conclusion and Future Work

The present study was designed to deal with the reliability and fault tolerance issue in SDN. In particular, the paper studies the reliable and fault tolerant path selection problem and propose an algorithm that enables the selection of a reliable shortest path towards fast reconfiguring a data plane after links failure. The first experimental results showed how the installation time of path's rules is directly proportional to the length of the path itself, then we defined the relation between the number of rules and edges in each path to be the foundation of our proposal method. Although the selected primary and backup paths have the same length, we have shown how the time reduction can be achieved with our proposed algorithm. Finally, we developed our algorithm to be as a POX controller component as well as evaluated the prototype of the developed component using the Mininet emulator. To the best of our knowledge, this is the first paper considering the reliability as a maximum shared of edges for SDNs.

In future, we will expand the work by investigating some real world network topologies like the European Reference Network and we also plan to consider definitions of reliability that are quantitative, for example, combining it with risk measures and metrics in order to obtain a more precise reliable network topology configuration.

ACKNOWLEDGEMENTS

This work is supported by the College of Computer Science and Information Technology at the University of Al-Qadisiyah under the scholarship program of the Iraqi Ministry of Higher Education and Scientific Research.

References

[1] Farhady, H., Lee, H., & Nakao, A. (2015). Software-defined networking: A survey. *Computer Networks, 81*, 79-95.

[2] Lin, P., Bi, J., Hu, H., Feng, T., & Jiang, X. (2011, November). A quick survey on selected approaches for preparing programmable networks. In *Proceedings of the 7th Asian Internet Engineering Conference* (pp. 160-163). ACM.

[3] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., & Shenker, S. (2007, August). Ethane: taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review* (Vol. 37, No. 4, pp. 1-12). ACM.

[4] Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2016). Research challenges for traffic engineering in software defined networks. *IEEE Network, 30*(3), 52-58.

[5] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review, 38*(2), 69-74.

[6] Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks, 71*, 1-30.

[7] Sharma, S., Staessens, D., Colle, D., Pickavet, M., & Demeester, P. (2013). OpenFlow: Meeting carrier-grade recovery requirements. *Computer Communications, 36*(6), 656-665.

[8] Sgambelluri, A., Giorgetti, A., Cugini, F., Paolucci, F., & Castoldi, P. (2013). OpenFlow-based segment protection in Ethernet networks. *Journal of Optical Communications and Networking, 5*(9), 1066-1075.

[9] Luo, M., Zeng, Y., Li, J., & Chou, W. (2015). An adaptive multi-path computation framework for centrally controlled networks. *Computer Networks, 83*, 30-44.

[10] Kim, H., Schlansker, M., Santos, J. R., Tourrilhes, J., Turner, Y., & Feamster, N. (2012, October). Coronet: Fault tolerance for software defined networks. In *2012 20th IEEE International Conference on Network Protocols (ICNP)* (pp. 1-2). IEEE.

[11] Astaneh, S. A., & Heydari, S. S. (2016). Optimization of SDN flow operations in multi-failure restoration scenarios. *IEEE Transactions on Network and Service Management, 13*(3), 421-432.

[12] Lantz, B., Heller, B., & McKeown, N. (2010, October). A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (p. 19). ACM.

[13] POX , https://openflow.stanford.edu/display/ONL/POX+Wiki

[14] Desai, M., & Nandagopal, T. (2010, January). Coping with link failures in centralized control plane architectures. In *2010 Second International Conference on COMmunication Systems and NETworks (COMSNETS 2010)* (pp. 1-10). IEEE.

[15] Dasgupta, M., & Biswas, G. P. (2012). Design of multi-path data routing algorithm based on network reliability. *Computers & Electrical Engineering, 38*(6), 1433-1443.

[16] Upadhaya, S., & Devi, G. (2010). Characterization of QoS based routing algorithms. *International Journal of Computer Science & Emerging Technologies, 133*.

[17] Kelly, F., & Voice, T. (2005). Stability of end-to-end algorithms for joint routing and rate control. *ACM SIGCOMM Computer Communication Review, 35*(2), 5-12.

[18] Han, H., Shakkottai, S., Hollot, C. V., Srikant, R., & Towsley, D. (2006). Multi-path tcp: a joint congestion control and routing scheme to exploit path diversity in the internzet. *IEEE/ACM Transactions on Networking (TON), 14*(6), 1260-1271.

[19] Banerjee, S., Ghosh, R. K., & Reddy, A. P. K. (1996). Parallel algorithm for shortest pairs of edge-disjoint paths. *Journal of parallel and distributed computing, 33*(2), 165-171.

[20] Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *management Science, 17*(11), 712-716.

[21] Guo, Y., Kuipers, F., & Van Mieghem, P. (2003). Link-disjoint paths for reliable QoS routing. *International Journal of Communication Systems, 16*(9), 779-798.

[22] Schult, D. A., & Swart, P. (2008). Exploring network structure, *dynamics*, and function using NetworkX. In *Proceedings of the 7th Python in Science Conferences (SciPy 2008)* (Vol. 2008, pp. 11-16).