# VIRTUAL TOPOLOGY PARTITIONING TOWARDS AN EFFICIENT FAILURE RECOVERY OF SOFTWARE DEFINED NETWORKS

**ALI MALIK[1], BENJAMIN AZIZ[1], CHIH-HENG KE[2], HAN LIU[1], MO ADDA[1]**

[1]School of Computing University of Portsmouth, Portsmouth PO1 3HE, UK
[2]Department of Computer Science and Information Engineering, National Quemoy University, Taiwan
E-MAIL: {ali.al-bdairi, benjamin.aziz, han.liu, mo.adda}@port.ac.uk, smallko@gmail.com

**Abstract:**

**Software Defined Networking is a new networking paradigm that has emerged recently as a promising solution for tackling the inflexibility of the classical IP networks. The centralized approach of SDN yields a broad area for intelligence to optimise the network at various levels. Fault tolerance is considered one of the most current research challenges that facing the SDN, hence, in this paper we introduce a new method that computes an alternative paths reactively for centrally controlled networks like SDN. The proposed method aims to reduce the update operation cost that the SDN network controller would spend in order to recover from a single link failure. Through utilising the principle of community detection, we define a new network model for the sake of improving the network's fault tolerance capability. An experimental study is reported showing the performance of the proposed method. Based on the results, some further directions are suggested in the context of machine learning towards achieving further advances in this research area.**

**Keywords:**

**Network Topology, Community Detection, Graph Theory, Software Defined Networking**

## 1. Introduction

The Internet and networking system in general plays an essential role in changing our life style through producing various type of technologies that get involved in our daily activities such as social media, economics and business. Networking devices exchange data in a variety networks (e.g. wireless sensor networks, the Internet-of-Things and Cloud networks) where currently there are about 9 billion devices connected to the Internet and this number is expected to more than double by 2020. However, the today's IP network infrastructures do not have the ability to accommodate such a huge number of devices, hence the Internet ossification is highly expected [1]. One effective solution to tackle the ossification issue is via replace the complex and rigid networking systems by a programmable networking instead.

Software-defined networking (SDN) was resulted from a long history of efforts that have been exerted for the purpose of simplifying the computer networks management and control [2]. In SDN the *control plane* has been moved out from the *data plane* and placed in a central location usually called the network *controller* or the network operating system. Due to this decoupling, the network forwarding elements (e.g. routers and switches) became a dummy devices, which typically dictated by the network controller. In reality, the SDN controller is either has a direct connection to the data plane forwarding elements or indirect way through which the controller will be able to instruct those devices. The OpenFlow [3] protocol is the most commonly used to establish the connection between the data plane forwarding elements and the controller, so the controller will be able to send the *forwarding rules* to each device that lies on its domain. So far, SDN has gained much attention of both academia and industry community and adopted by some of well known pioneering companies like Deutsche Telekom, Google, Microsoft, Verizon, and Yahoo [4]. Although SDN has brought a significant benefits to the concept of networks, some new challenges accompanied this innovation such as faults, configuration, security and performance [5]. Thus, our goal in this paper is to eliminate the SDN drawbacks through mitigating the one of the most current challenges, which is the data plane fault tolerance.

The remainder of this paper is organised as follows. In Section 2 various techniques to solve the issue of SDN data plane failures are presented. Section 3 and 4 illustrate our model and the proposed framework. The experimental results of the evaluation are presented in section 5. Finally, the conclusion of this paper and future work directions are provided in section 6.

## 2. Related work

The topic of SDN faults and recovery has been already studied, so we will discuss the relevant literature in this section. Since the SDN has two separated planes (i.e. control and data), hence each plane is susceptible to failure. Apart from control plane failure, which is not the main focused in this work, the recovery mechanism of data plane can be classified into *protection* and *restoration* techniques [6]. In protection method, which also known as *proactive*, the flow entries of the backup paths are determined in early stage of failure, hence the effected data packets will convey through the backup path directly at the moment of failure. On the other hand, in restoration mechanism, which also known as *reactive*, the backup path will be determined by the controller after the occurrence of failure, thus extra time is required to set up the discovered alternative path. Authors in [7] and [8] have shown how a fast data plane recovery can be achieved within the protection approach. However, the cost of proactive mechanisms is high as it consumes the capacity of the Ternary Content Addressable Memory (TCAM) [6], where the forwarding rules to be stored. In addition, there is no guarantee that the pre-planned paths will be in a good health once the primary paths will fail.

In contrast, efforts have been exerted to investigate the effectiveness of the reactive approach. In this context, the authors in [9] and [10] have shown how a fast restoration can be accomplished, however in both works the experimental topologies were small scale (i.e. 6 and 14 nodes respectively). Furthermore, the processing time for setting up the chosen path was ignored, which is a requirement in SDNs in order to re-routing from the affected primary path to the backup one.

Unlike the previous studies, the authors in [11] produced a new method for fast restoration by reducing the processing time, which typically paid by the controller, through finding the alternative path (from *end-to-end*) that has a minimum operation requirement. One main drawback with this work is that it does not guarantee the sequentiality of the health nodes in the effected path to be in similar order on the alternative one. Additionally, there is a lack of information regarding the simulation tool that has been used. We also noted that there was a mistake with the first network topology that used in their experiments, since the authors mentioned that it contains 37 nodes and 57 links, while in reality their network contains a 56 links as it missed the edge that connects between Hamburg and Frankfurt according to the SNDlib library [12], hence their results could be significantly deviant from the expected one.

All the above issues motivated us to put a further efforts in order to investigate a more possible solutions for the problem of SDN fault tolerance. The next section will present the proposed system model of this work.

## 3. System model

The most frequently used notations throughout this paper are listed in Table 1.

**TABLE 1.** List of notations

| Symbol | Description |
|---|---|
| $C$ | The set of cliques |
| $c$ | The failed clique |
| $r_s$ | Source router |
| $r_d$ | Destination router |
| $r_{ic}$ | Any arbitrary router in the failed clique |
| $r_{jc}$ | Any arbitrary router in the failed clique |
| $P_{min}$ | Dijkstra's shortest path in terms of number of hops |
| $D_G$ | Dijkstra for finding the shortest path based on the graph $G$ |
| $D_c$ | Dijkstra for finding the shortest path based on the clique $c$ |

We utilised the *undirected graph* theory as a basis to model the computer network topology. Generally, each simple graph $G = (V, E)$ consists of a set of vertices (i.e. nodes), $V$, as well as a set of edges (i.e. links), $E$, which connect the nodes to one another. The set of all links in $G$ can be defined as a 2-element subset of nodes, $E \subseteq V \times V$. We define a path $P$, from source to destination, as a *sequence* of consecutive vertices representing nodes or routers in the network[1]. The path starts at the source router, $r_s$, and ends with a destination router, $r_d$, with $r_{ic}$ and $r_{jc}$ being any two adjacent routers along $P$:

$$P = (r_s, \ldots, r_{ic}, r_{jc}, \ldots, r_d)$$

We define the set of all possible paths, $\mathcal{P}_{r_s,r_d}$, between any source router $r_s$ and destination router $r_d$, as the following set:

$$\mathcal{P}_{r_s,r_d} = \{P \mid (\mathit{first}(P) = r_s) \wedge (\mathit{last}(P) = r_d)\}$$

and the definition of *first* and *last* is given as functions on any general sequence $(a_1, \ldots, a_n)$:

$$\mathit{first}((a_1, \ldots, a_n)) = a_1$$
$$\mathit{last}((a_1, \ldots, a_n)) = a_n$$

[1] We use the terms router and node interchangeably.

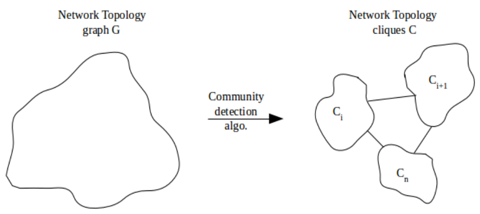

**FIGURE 1.** Community detection

Community detection (or sometimes known as cliques identification) has been productively proposed as a solution to resolve various kinds of network-related problems including the problem of network path optimisation. In this context, we will use the concept of non-overlapping *cliques* as an approach to optimise SDN restoration through accelerating the process of failure recovery. By dividing the network's graph $G$ into a certain number of cliques, we are assuming that when the link failure occurs then only one clique will suffer from the failure. Meantime, the other cliques should be working fine. It is highly likely that most of the path's links will be distributed over various cliques, hence at the link failure moment, the only clique $c$, which includes the effected link, will be treated rather than dealing with the whole graph $G$ in looking for an alternative path from *end-to-end*.

Informally, Figure 1 depicts how the virtual cliques can be extracted from the original network topology graph through applying any community detection algorithm, where the number of the extracted cliques is vary and typically depends on the network topology and the used algorithm itself. In our model, we define the set of cliques dividing the network graph $G$ as $C \subseteq G$, where individual cliques $c \in C$ are defined as follows:

$$c = (V', E') \mid V' \subseteq V \wedge E' \subseteq E$$

The definition of cliques is mutually exclusive, in other words:

$$\forall c_1, c_2 \in C \mid c_1 = (V_1, E_1) \ \wedge \ c_2 = (V_2, E_2) \ \Rightarrow \ (V_1 \cap V_2 = \emptyset) \ \wedge \ (E_1 \cap E_2 = \emptyset)$$

We define a failed link (i.e. a 2-router sub-path), $F$, in a path $P$ as follows:

$$F = (r_{ic},\ r_{jc}) \mid \exists\, c : c = (V', E') \wedge F \in E'$$

This definition of $F$ assumes only those cases where failed links are always of an intra-clique type, i.e. it will never be the case that the failed link connects two cliques, i.e.:

$$\nexists F = (r_{ic},\ r_{jc}), c_1 = (V_1, E_1), c_2 = (V_2, E_2) \mid c_1 \neq c_2 \ \Rightarrow \ r_{ic} \in V_1 \ \wedge \ r_{jc} \in V_2$$

Inter-clique failures will be the focus of future work.

We use the term *longest-shortest path* as the path that has the maximum number of hops amongst the set of solutions returned by applying Dijkstra's algorithm [13] to our network topology for finding the shortest path between every possible two nodes in that network. To find this longest-shortest path, we define the special function *LS* as follows:

$$LS(\mathcal{P}_{Dset}) = x,\ \textit{such that}\ x \in \mathcal{P}_{Dset}\ \textit{and}\ \forall y \in \mathcal{P}_{Dset} : len(y) \leq len(x)$$

If there are more than one longest-shortest paths in $\mathcal{P}_{Dset}$, we pick one randomly. $\mathcal{P}_{Dset}$ itself represents the set of all Dijkstra-based solutions for some network topology $(V, E)$:

$$\mathcal{P}_{Dset} = \{P \mid \forall\, r_s, r_d \in V : P = D(\mathcal{P}_{r_s, r_d})\}$$

## 4. The Proposed Framework

From a high level view, Figure 2 illustrates the main components of our proposed framework where the Fault Tolerance Enhancer component is the primary contribution of this framework. We discuss next in more detail the developed component as well as the tools that we used in this framework.

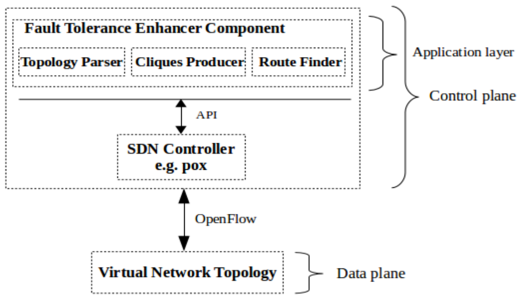

**FIGURE 2.** Proposed framework components

### 4.1 SDN controller

SDN controller representing the network's brain and the most vital part, which is the place where the intelligence and decision making resides. Presently, there are more than 30 different SDN controller offering from both academia (e.g. for research purposes) and industry (e.g. for commercial use). Our framework currently supports the POX controller [14], which is a python-based open source SDN controller. We selected the POX as it is more suitable for research purposes and also more convenient for fast prototyping than the other available ones [15]. The OpenFlow [3] protocol is used to communicate between the control and data planes in order to gather statistics from the data plane and carry the controller instructions to program the data plane elements, whereas the set of POX APIs can be utilised for developing various network applications.

### 4.2 Fault tolerance enhancer component

Currently, there are three main parts that consisting this component as follows:
**A**. *Topology parser*: will be responsible for fetching the underlying network topology characteristics and build a topological view with the aid of the POX *openflow.discovery*, which is an already developed component. In order to represent the gained network topology as a graph $G$, we utilised the NetworkX [16] tool, which is a pure python package with a set of functions that can be used to manipulate and simplify the network graphs.

**B**. *Cliques producer*: is responsible for virtually partitioning the network topology graph $G$ into $C$ (i.e. sub-graphs) by incorporating the well known community detection algorithm *Girvan and Newman* [17] to produce the possible cliques (with any size) on the basis of the network graph that acquired from the topology parser. The densely connection between the resulted clique's vertices is the main feature of Girvan and Newman that interesting to us, in other words, the strong connection among the nodes in each clique could provide a multiple alternative paths that would utilised at the failure events. To do so, we have incorporated the *igraph* tool [18] to our framework, which is a python open source library for analysing and manipulating the graphs where the Girvan and Newman algorithm is already implemented.

**C**. *Route finder*: is used to identify the paths for the network's flows on the basis of the global view of the underlay topology, which can be gained from the topology parser. in fact, the route finder will be called on two occasions; the first one when a new packet arrives to the network, while the second one when the failure occurs so a new path should be computed. For this purpose, two algorithms have been developed to obtain the shortest path based on the well known Dijkstra's algorithm. We adopt the hop count as an additive metric on which the Dijkstra's algorithm will be able to form the shortest path for any incoming requests. The first algorithm reflects the default action that performed by the SDN controller at failure moments, which is to erase the flow entries of the effected path and then install the rules of the backup one from the source router $r_s$ to destination $r_d$, the pseudo code is demonstrated in Algorithm 1.

**Algorithm 1:** First algorithm to find the shortest path with Dijkstra from End-to-End based on the Graph $G$

```
  On Normal: Set Primary Path as $P_{min} \in \mathcal{P}_{r_s,r_d}$
  On Failure : Do the following procedure
1 $\mathcal{P}_{r_s,r_d} := \mathcal{P}_{r_s,r_d} - \{P_{min}\}$
2 $P_{min} := D_G(\mathcal{P}_{r_s,r_d})$
```

On the other hand, the second algorithm will tackle the failure based on the effected clique $c \in C$ rather than searching from *end-to-end* based on the whole graph $G$, the pseudo code of this algorithm is illustrated in Algorithm 2. Currently, the Algorithm 2 is capable to manage the clique's intra-link failure and for the future we will extend the framework to include the inter-link failure among the cliques.

**Algorithm 2:** Second algorithm to find the shortest path with Dijkstra on the basis of either $G$ or effected $c$

```
  On Normal: Set Primary Path as $P_{min} \in \mathcal{P}_{r_s,r_d}$
  On Failure : Do the following procedure
1 if $F = (r_{ic}, r_{jc})$ then
2     $P_{min} := D_c(\mathcal{P}_{r_{ic},r_{jc}})$
3 end
```

The complexity of the two above algorithms are similar to the Dijkstra, which is $O(|V|+|E|\ log\ |V|)$. It also worth noting here that both algorithms have the same strategy to obtain the shortest path for the newly arriving packets, which is by running the Dijkstra based on the network graph $G$. However, they differ in respect of finding the alternative path to recover from the failure, in which the Algorithm 2 will turn to apply the Dijkstra on the $c$ rather than $G$.

## 5. Simulations and experimental results

In order to test the performance of the proposed framework, we first build it on top of POX controller as illustrated in Figure 2, then we used the popular SDN emulator Mininet [19] to evaluate the prototype of our proposed framework. As mentioned earlier that SDN has splitted the network architecture into control plane, the brain, and data plane, the muscles. So, we have

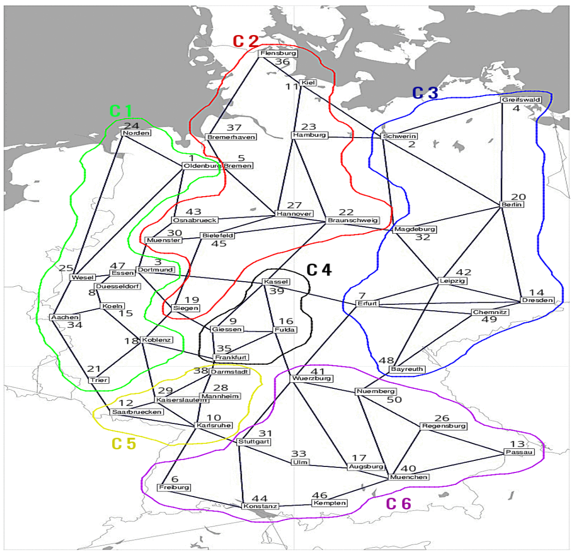

**FIGURE 3.** Cliques-based network topology of Germany50, [12]

modelled the Germany50 from SNDlib [12] as an instance of a real world network topology to act our data plane structure, which contains 50 nodes and 88 edges. The black links in Figure 3 represent the original Germany50 topology and we added the colored layer (i.e. from `c1` to `c6`) to demonstrate the cliques that achieved after applying the Girvan and Newman algorithm. According to the Germany50 topology, the longest-shortest path lies between 1 and 13 through the route:(*1, 5, 27, 22, 32, 42, 48, 50, 26, 13*), there might be several longest-shortest paths in the network topology and in such a case we choose randomly. We attached two virtual hosts $H_1$ and $H_2$ to the $r_s$ and $r_d$ of the longest-shortest path respectively in order to simulate the scenario of packet injection. For now let's assume that the link (*27, 22*) fails. Consequently, Algorithm 1 will response to the failure by removing the failed-primary path and install (*1, 25, 47, 3, 39, 7, 41, 17, 40, 13*) as an alternative shortest path to mask the failure. Since our selected path passes through 4 cliques, which are `c1`, `c2`, `c3` and `c6`, this means only one clique (e.g. sub-graph) will be involved at the moment of the intra link failure in the scenario of Algorithm 2, which will react first by detecting the effected clique $c$ (i.e. `c2` in our case) and thence find a loop-free shortest path between the routers on both sides of the failed link, i.e. between the two nodes of $F$. According to the given example, the sub-path (*27, 45, 22*) will be returned by the second algorithm as a quick solution based on `c2` without taking into consideration the other clique's nodes as they will remain on the same settings (untouched). Hence, it can be clearly seen that the second algorithm does not guarantee the shortest path from *end-to-end* as it works over a clique and not a whole graph. We are expecting a significant reduction in the duration of the recovery process for two reasons, (I) because the proposed algorithm will consider only one clique but not the whole graph and (II) due to minimising the number of rules modification. Figure 4 shows the behaviour of the two algorithms, which is based on the selected path (i.e. from 1 to 13), in both failure and non-failure scenarios. We note that the two algorithms have nearly similar

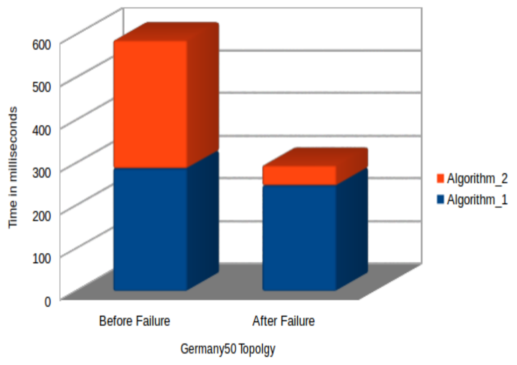

**FIGURE 4.** Before and after failure measurements

durations for setting up the path (i.e. before failure scenario). However unlike the Algorithm 1, Algorithm 2 has an extremely positive impact in terms of enhancing the time cost of fault tolerance. The reduction rate can be calculated through comparing the consumed time before and after the failure, which can be arrived through the following formula:

$$R = \frac{\mathit{Time\_BF} - \mathit{Time\_AF}}{\mathit{Time\_BF}} \times 100 \qquad (1)$$

Where $R$ represents the reduction rate, *Time_BF* is the time of setting up the primary path before the failure and *Time_AF* is the time of setting up the alternative path after the failure. Based on (1) and according to the result of figure 4, the $R$ of Algorithm 2 is 84.333%. Let us assume that $len(r_s, ..., r_d) = n$, then the utilisation rate of Algorithm 2 can be measured through $(\frac{n-2}{n}) \times 100\%$, hence, the pre-installed rules utilisation percentage (based on our example case) will be 80%, this ratio interprets the high reduction rate that gained by the algorithm. As a result, the larger the value of $n$, the more utilisation we obtain out of the selected primary path.

## 6. Conclusions

This paper provides a new approach for ameliorating the fault tolerance mechanism of software-defined networks. We have defined a new network model based on the set theory and graph theory to represent the theoretical part. We also have developed a new framework on the basis of the created model to represent the practical side of this work. The experimental results, which have been carried out through a well-known tools and emulation, demonstrate how the proposed method has led to improving the performance of reactive failure recovery through segmenting the network topology into a certain number of non-overlapping cliques. The proposed approach has not been explored so far, which indicates that this paper first time suggests the partitioning as a technique towards enhancing the restoration mode of the SDN's fault tolerance.

In future, we will position the study in the setting of machine learning, towards achieving that the routing solution can be dynamically adjusted according to the update of the statistical details of topological data. In other words, we aim to involve learning strategies in routing to achieve globally optimal search towards finding the shortest path.

## 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] 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.

[2] Feamster, N., Rexford, J., & Zegura, E. (2014). The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review, 44*(2), 87-98.

[3] 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.

[4] Kreutz, D., Ramos, F. M., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE, 103* (1), 14-76.

[5] Wickboldt, J. A., De Jesus, W. P., Isolani, P. H., Both, C. B., Rochol, J., & Granville, L. Z. (2015). Software-defined networking: management requirements and challenges. *IEEE Communications Magazine, 53*(1), 278-285.

[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] Kempf, J., Bellagamba, E., Kern, A., Jocha, D., Takcs, A., & Skldstrm, P. (2012, June). Scalable fault management for OpenFlow. In *Communications (ICC), 2012 IEEE International Conference* on (pp. 6606-6610). IEEE.

[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] Sharma, S., Staessens, D., Colle, D., Pickavet, M., & Demeester, P. (2011, October). Enabling fast failure recovery in OpenFlow networks. In *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the* (pp. 164-171). IEEE.

[10] Staessens, D., Sharma, S., Colle, D., Pickavet, M., & Demeester, P. (2011, October). Software defined networking: Meeting carrier grade requirements. In *Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on* (pp. 1-6). 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] SNDlib library, http://sndlib.zib.de.

[13] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik, 1*(1), 269-271.

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

[15] Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., & Smeliansky, R. (2013, October). Advanced study of SDN/OpenFlow controllers. In *Proceedings of the 9th central & eastern european software engineering conference in russia* (p. 1). ACM.

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

[17] Girvan, M., & Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the national academy of sciences, 99*(12), 7821-7826.

[18] Csardi, G., & Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal, Complex Systems, 1695*(5), 1-9.

[19] 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.