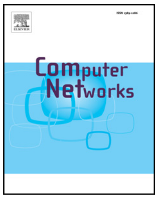
Computer Networks 170 (2020) 107104

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Smart routing: Towards proactive fault handling of software-defined networks

Ali Malik[a,*], Benjamin Aziz[b], Mo Adda[b], Chih-Heng Ke[c]

[a] School of Electrical and Electronic Engineering, Technological University Dublin, Dublin D08 NF82, Ireland
[b] School of Computing, Buckingham Building, University of Portsmouth, Portsmouth PO1 3HE, United Kingdom
[c] Department of Computer Science and Information Engineering, National Quemoy University, Kinmen 892, Taiwan

ARTICLE INFO

Article history:
Received 8 January 2019
Revised 26 December 2019
Accepted 13 January 2020
Available online 16 January 2020

Keywords:
OpenFlow
Software-defined networking
Fault management
Risk management
Service availability

ABSTRACT

In recent years, the emerging paradigm of software-defined networking has become a hot and thriving topic in both the industrial and academic sectors. Software-defined networking offers numerous benefits against legacy networking systems by simplifying the process of network management through reducing the cost of network configurations. Currently, data plane fault management is limited to two mechanisms: *proactive* and *reactive*. These fault management and recovery techniques are activated only after a failure occurrence and hence packet loss is highly likely to occur. This is due to convergence time where new network paths will need to be allocated in order to forward the affected traffic rather than drop it. Such convergence leads to temporary service disruption and unavailability. Practically, not only the speed of recovery mechanisms affects the convergence, but also the delay caused by the process of failure detection. In this paper, we define a new approach for data plane fault management in software-defined networks where the goal is to eliminate the convergence process altogether rather than accelerate the failure detection and recovery. We propose a new framework, called *Smart Routing*, which allows the network controller to receive forewarning signs on failures and hence avoid risky paths before the failure incidents occur. The proposed approach aims to decrease service disruption, which in turn increases network service availability. We validate our framework through a set of experiments that demonstrate how the underlying model runs and its impact on improving service availability. We take as example of the applicability of the new framework three types of topologies covering real and simulated networks.

1. Introduction

The concern about the Internet ossification, which is a consequence of the growing number of variety networks (e.g. IoT, WSN, Cloud, etc.) that serve up to 9 billion users around the globe, has led to investigate for replacing the existing rigid network infrastructure with programmable one [1]. In this context, Software-Defined Networking (SDN) has been emerged as a promising solution for tackling the inflexibility of the legacy networking systems. In fact, SDN is part of a long history of attempts that aim to lower the barrier of deploying new innovations and make the network more programmable. For more on the history of programmable networks, we refer the interested readers to [2]. Unlike traditional IP networks, SDN architectures consist of three planes: *control*, *data* and *application*. The *control plane*, or sometimes called the *controller*, represents the network brain, which provides the essential functions and exerts a granular control by relying on the global view over the network topology, which is a crucial feature that has been missed in the past. The *data plane* comprises network forwarding elements that constitute the network topology. These forwarding elements are dictated by the network controller and therefore the entire nodes need to disclose their status periodically to the controller, hence the global view comes. In general, many studies have classified the SDN into two layers by considering the *application plane* as a complementary part to the control layer to solve various kinds of network issues such as firewall and load balance. So far, OpenFlow [3] is the most widely used protocol that enables the controller to govern the SDN data plane through carrying the *forwarding rules* as well as to capacitate the exchanging of signals between the two planes. Recently, SDN was not solely confined to the academic field but also gained the attraction of industry such as Google and Microsoft [4].

---

[*] Corresponding author.

E-mail addresses: ali.malik@tudublin.ie, up714266@myport.ac.uk (A. Malik), benjamin.aziz@port.ac.uk (B. Aziz), mo.adda@port.ac.uk (M. Adda), smallko@gmail.com (C.-H. Ke).

https://doi.org/10.1016/j.comnet.2020.107104

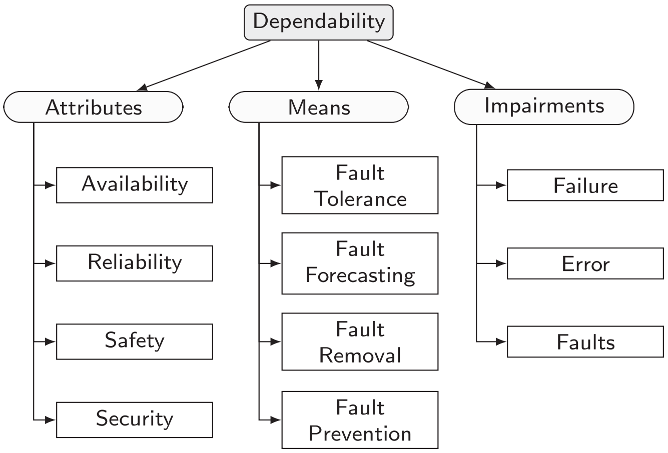


Fig. 1. The compartmentalisation of dependability concept, Laprie [5].

Nowadays, communication networks play a vital role in human being's life activities as it represents the backbone for the modern technologies. Networking equipment are failure prone and therefore, some aspects like availability, reliability and fault management are necessary. The term *dependability* is the umbrella that encompasses the aforementioned aspects and Fig. 1 gives an overall picture on the dependability taxonomy according to [5]. However, this paper focuses on the availability attribute by means of fault tolerance and forecasting of SDN link failures.

Although SDNs have brought a bunch of benefits with significant network improvements, some new challenges that accompanied with this innovation, such as security [6,7] and recovery from failure, still need to be addressed thoroughly in order to maximise the utility of SDNs [8,9]. Data plane link failure is considered to be a manifold problem. This is because the controller needs first to be notified about the failure and then to compute alternative routes in order to update the affected forwarding elements. The issue of network link failures is not a recent phenomenon, as it takes place in everyday operation with variations in living-time and causes [10]. However, the new architecture of OpenFlow requires more investigation in order to eliminate the challenges that hamper its growth.

In order to maximise the service availability in SDNs, we define a new approach that minimises the percentage of service unavailability by using online failure prediction. This allows the network controller to perform the necessary reconfiguration prior to the occurrence of failure incidents. Although a number of works on SDN fault management have been proposed, none of them has exploited the global view of SDNs in the context of failure prediction. With this context in mind, we can summarise the main contributions of this paper as follows:

- A new network model that allows for the forecasting of link failures by predicting their characteristics in an online fashion. This model also demonstrates how links failure prediction can be integrated into the process of proactive restoration with the aid of risk analysis.
- We provide an implementation of the new model in terms of a couple of fault tolerance algorithms. We use simulation techniques to test the efficiency of these algorithms. Our simulation results prove that the proposed model and algorithms improve the service availability of SDNs.

The rest of the paper is organised as follows. Section 2 introduces various SDN fault management techniques from the literature. The problem statement is highlighted in Section 3. We then present our network model and framework in Section 4. Sections 7 and 8 present the experimental procedure, observed result and comparison. Finally, the summary of this paper is provided in Section 9 with some future directions.

2. Related work

Link failure issue often occurs as a part of everyday operation. Due its importance and the negative impact it has on the network Quality of Service (QoS), a considerable amount of research has been conducted to analyse, characterise, evaluate and recover from the frequent issue of link failure. While, the physical separation of the control plane from the data plane results into two independent entities. Both entities are susceptible to failure incidents. According to [11], control plane failures are more severe than other failures. This is due to the significant role of network controller in managing the whole network activities. For more details about control plane failures, we refer the interested readers to [12,13]. However, in this paper, we are focusing on the data plane link failures only.

Communication networks are prone to either unintentional failures, *unplanned*, due to various causes such as human errors, natural disasters like earthquakes, overload, software bugs, cable cut and so on, or to intentional failures, *planned*, that caused by the process of maintenance [14,15]. Failure recovery scheme is a necessary requirement for networking system to ensure the reliability and service availability. Generally, failure recovery mechanisms of carrier-grade networks are categorized into two types: *proactive* and *reactive*. In case of link failure, resilience mechanism of SDNs ought to redirect the affected flows in order to avoid the location of failure and keep the system continue working despite the presence of the abnormal situation. SDN controller has the ability to mask the data plane failure either proactively or reactively [16], while each of them has pros and cons. In this section, we discuss current efforts to tackle the data plane link failures.

2.1. Proactive

In proactive, which is also know as *protection*, the alternative paths are preplanned and reserved in advance (before a failure occurs). According to [17], there are three protection schemes that can be applied to recovery from network failure:

- One to One (1: 1): In which, one protection path is dedicated to protect exactly one path.
- One to Many (1: Y): In which, one protection path is dedicated to protect up to Y paths.
- Many to Many (X: Y): In which, X of specified protection paths are dedicated to protect up to Y working paths such that $X \leq Y$.

The authors in [18] have implemented an OpenFlow monitoring function for achieving a fast data plane recovery. In [19], another protection method has been proposed through using the OpenFlow-based Segment Protection (OSP) scheme. The main disadvantage of this strategy is that it consumes the data plane storing capability since the more flow entries (i.e rules) to be stored the more space will be used, however, the current OpenFlow appliances in the market are able to accommodate up to 8000 flow entries due to the limitation of the Ternary Content-Addressable Memory (TCAM), thence such a kind of solution is costly [4,16], where this issue was discussed in some studies such as [20,21]. In addition, the installation of many attributes in the OpenFlow forwarding elements could lead to deteriorate the process of match-and-action for the data plane nodes. Moreover, there is no guarantee that the preserved backups are failure-free, in other words, the backup path might fail before the primary one and resulting a waste in space and time.

2.2. Reactive

In reactive, which is also called *restoration*, the possible alternative paths are not preplanned and will be calculated dynamically when failure occurs. The authors in [22,23] presented an OpenFlow restoration method to recover from single-link failures. However, their experiments were only conducted on small scale network topologies not exceeding 14 nodes. In [24], the authors have demonstrated through extensive experiments that OpenFlow restoration is not easily attainable within 50ms, especially for large scale networks, unless using the protection technique.

In the same context, some works have utilised the concept of multiple disjoint paths to be employed as a backup. For example, CORONET [25] is presented as a fault tolerance system for SDNs, in which multiple link failures can be resolved. The ADaptive Multi-Path Computation Framework (ADMPCF) [26] for large scale OpenFlow networks is produced as a traffic engineering tool that capable to hold two or more disjoint paths to be utilised when some network events occur (e.g. link failure). HiQoS [27] is introduced as a traffic engineering tool towards better QoS for SDNs. HiQoS computed multipath (at least two constrained paths) between all possible pairs in a network, hence a quick recovery from a link failure is attainable. Most of the existing works did not take into account the processing time of flow entries (e.g. insert, delete and modify) that need to be updated. Although the performance of OpenFlow devices are associated with their manufacturer specification, in [28], the authors stated that each single flow entry insertion is ranging from 0.5ms to 10ms. However, 11ms is the minimum duration that required to modify a single rule since each modification process includes both deletion (old rule) and insertion (new one) of rules [29]. There are a number of studies like [30–32] used the OpenState mechanism to recover from data plane failures without being dependant on the network controller and hence reducing the overload on controller and speedup the process of recovery. However, such approaches still inapplicable as the existing OpenFlow equipment does not support such customization.

Unlike the existing works, the authors in [33] have dealt with problem of minimising the time of flow entries that required to divert from an affected primary path to backup one. Although, the presented algorithms did not guarantee the shortest path from end-to-end, but it opens a new direction that worth to be explored. Within the same context, authors in [34] produced new algorithms for minimising the required time to update through reducing the solution search space from source to destination in the affected path. Similarly, in [35] an approach to divide the network topology into non-overlapping cliques has been produced to tackle the failure issue in local-based manner rather than global. Both [34,35] took into account the time required to compute the alternative route in order to speed up the operation of update. While, the main issue with the last three works is that it does not secure the shortest path from source to destination.

2.3. Summary

In summary, the previous studies produced different methods to tackle the problem of data plane recovery from link failure incidents and for more details about SDN fault management we refer the interested readers to the recent survey [11,36]. Eventually, protection techniques are not ideal due to the TCAM space exhaustion, whereas the latency issue is the major drawback of the existing restoration methods. As a result, SDN fault management still needs more research and investigation.

3. Problem statement

Distinctly, the existing SDN fault management techniques are getting involved after a failure occurrence. Thus, it cannot prevent a certain impact on traffic flows such as service unavailability. This problem occurs due to the delay of the convergence scheme T_C. We define T_C as the required time to amend a path in response to a failure scenario. Typically, the convergence time in SDN can be summarised as a combination of three factors:

- *Failure detection time* (T_D): This is the time required to detect a failure incident. Comparing with the conventional networking systems, the centralised management and global view of SDN ease this task by continuously monitoring the network status and get notifications upon failures. However, the speed of receiving a notification is sometimes associated with the nature of network design and mode of communication (in-band or out-of-band) [37,38]. According to [39], the link failure detection time is ranging from tens to hundreds milliseconds, which may also rely on the type of commercial OpenFlow switch.
- *New route computation time* (T_{SP}): This is the spent time when network controller runs a nominated shortest path routing algorithm (e.g. Dijkstra [40]) to compute the backup path (usually for the reactive fault tolerance strategies). The T_{SP} computation time could reach 10s of milliseconds [34] according to how big the network is.
- *Flow entries update time* (T_{Update}): This is the time required to update relevant switches, i.e., nodes involved in the affected path. Again, this factor depends on how many forwarding rules need to be updated after the failure scenario where the amount of time for every single rule could exceed 10 ms.

Accordingly, the resulting convergence time can be calculated through the following equation:

$$T_C = T_D + T_{SP} + \sum_{src}^{dst} T_{Update} \tag{1}$$

Currently, the classical SDN fault management methods aim to tackle the failure after its occurrence, therefore, the recovery mechanism is activated after the moment of failure and hence all the previous work proposals embroiled in a certain amount of delay according to (1). The only way to completely overcome the three factors of (1) altogether is by handling the failure before it occurs. Therefore, failure prediction is required to provide awareness about the potential future incidents as well as allowing the controller to perform the reconfiguration action in purpose of overriding failures before causing damage on some paths. Although there are a number of studies have put efforts on the area of failure prediction, none of the traditional (except the work in [41]) and/or the new generation networking systems has exploited the information that can be gained from any prediction method to eliminate the network incidents (e.g. link failures). To the best of our knowledge, Vidalenc et al. [41] is the only realistic study that discussed the advantages of failure prediction through producing a risk-aware routing method for the legacy IP networks. While, our work is differ from them by building a realistic framework of proactive failure management for SDNs. In this work, we combine the concept of the online failure prediction with risk analysis towards maximising the network service availability.

4. The proposed model

Anticipating failures before they occur is a promising approach for further enhancement of an SDN failure management techniques, i.e., the proactive and reactive, in which the controller responds to failures when they take place. The SDN proposed model

Table 1
List of notations.

Symbol	Description
src	Source router
dst	Destination router
A	Service availability
U	Service unavailability
e_{ij}	Link traversing any two arbitrary routers i and j
Q_{ptr}	A pointer that points to first e_{ij} in the Queue
F	Failed link set
F_R	Failed/affected route set
PF_L	Potential failed link set
PF_R	Potential failed route set
M	Prediction alarm message
CO	Network controller
T_Ω	Threshold of failure probability
T_ω	Threshold of risk
OF	OpenFlow instruction
TP	True positive
FN	False negative
FP	False positive
CC	Cable cut per year
SP_x	Any shortest path algorithm x in terms of hops

for anticipating link failure events is presented in this section. We start first by outlining some of the notations we use in the rest of the paper, as shown in Table 1. The network topology is modelled as an *undirected graph* $G = (V, E)$; where V represents the finite set of vertices (i.e. routers) in G that ranges over by $\{v_i, v_j, \ldots, v_z\}$ where $\{i, j, \ldots, z\} \subset \{1, \ldots, n\}$ for $n \in \mathbb{N}$, and E represents the finite set of bidirectional edges (i.e. links) in G that denoted as $\{e_{ij}\}$ where each $e_{ij} \in E$ is an edge that enables v_i and v_j to connect each other. Now, we define the following test operational function (OP) over a link, which reflects the link state as follows:

$$OP(e_{ij}) = \begin{cases} 1 & \text{the link is operational} \\ 0 & \text{otherwise} \end{cases}$$

Therefore F can be defined as follows:

$$F = \{e_{ij} \mid e_{ij} \in E \wedge OP(e_{ij}) = 0\}$$

Based on G, we define a path P as a *sequence* of consecutive vertices representing routers in the network. Each path starts from a source router, src, and ends with a destination router, dst:

$$P = (src, \ldots, dst)$$

We define the set $Flow$ to represent all demand traffic flows that need to be serviced. Each $flow \in Flow$ is an instance of P, which associates with a particular traffic that are defined by unique src and dst. We consider $flow_{set}$ to be the set of all the possible paths between src and dst that can be derived from G, which is defined as follows:

$$flow_{set} = \{P \mid (first(P) = src) \wedge (last(P) = dst)\}$$

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

$$first((a_1, \ldots, a_n)) = a_1$$
$$last((a_1, \ldots, a_n)) = a_n$$

We also consider P_{set} as a set that contains all the admissible paths that can be constructed from G, this means $P \in P_{set}$ and therefore, $Flow \subset P_{set}$. When a link failure is reported in G, we identify the affected routes as follows:

$$F_R = \{flow \mid flow \in Flow \wedge \exists_{v_i, v_j}.v_i, v_j \in flow \wedge OP(v_i, v_j) = 0\}$$

In the same context, but this time we consider the case of when there is a link failure prediction message $m_i \in M$ such that M set denoted by $\{m_i\}_{i=1}^{n}$ where each $m_i \in M$ is defined as $m_i = (\bar{e}_{ij}, t)$, where t is the time when the system receives m_i. In this context, we define the following:

$$PF_L = \{\bar{e}_{ij} \mid \bar{e}_{ij} \in E \wedge \exists_{m_i}.m_i = (\bar{e}_{ij}, t) \wedge m_i \in M\}$$

to characterise the received link, which we use $\bar{e}_{ij}$ to imply that $e_{ij} \in PF_L$ is a shorthand, with state of *potential to fail* and hence it does not belong to F. Now, we can define the *potential to fail route* set as follows:

$$PF_R = \{\bar{flow} \mid \bar{flow} \in Flow \wedge (\exists_{\bar{e}_{ij}}.\bar{e}_{ij} \in \bar{flow} \wedge \bar{e}_{ij} \in PF_L)\}$$

where $\bar{flow}$ is a *flow* that has at least one $\bar{e}_{ij}$, in other words, $\bar{flow} \cap PF_L \neq \emptyset$.

4.1. SDN predictive model

All the previous efforts that dealt with data plane failures have succeeded in mitigating the impact of failures (e.g. reduce the downtime) rather than attempting to obviate their effect, such as minimise the service unavailability. Network incidents that cause routing instability, i.e., flaps, and lead to significant degrading of network service availability vary [42,43]. However, in our case, we are only concerned with data link failures. By relying on monitoring techniques, some failures can be predicted through failure tracking, syndrome monitoring, and error reporting [44]. Consequently, a set of conditions can be defined as a base to trigger a failure warning when at least one of the predefined conditions is satisfied. The following simple form illustrates rule-based failure prediction:

$$if\ \langle condition_1 \rangle\ then\ \langle warning\ trigger \rangle$$
$$\ldots$$
$$if\ \langle condition_n \rangle\ then\ \langle warning\ trigger \rangle$$

Online failure prediction strategies vary, such as machine learning techniques (e.g. using the κ-nearest neighbor algorithm [45]) and statistical analysis methods (e.g. time series [44], Kalman and Wiener filter [46]). Such techniques can be used to predict the incoming events in short-term based through relying on the past and current state information of a system. However, in this paper we do not intend to propose a failure prediction solution as extensive studies have been conducted in this field with remarkable achievements. Instead, employing the online failure prediction as a technique to enrich the current SDN fault management is one of the main aims of this work. A generic overview of the time relations of online failure prediction is presented in Fig. 2, which presumes the following:

- Δt_d: represents the historical data upon which the predictor is forecasting the upcoming failure events.
- Δt_l: represents the lead time, which is the time in which a failure alarm is generated. It can also be defined as the minimum duration between the prediction and failure.
- Δt_w: is the warning time in which an action may be required to find a new solution based on the predicted event. Therefore, Δt_l must be greater than Δt_w so that the information from prediction will be serviceable. In SDN, the Δt_w should be at least adequate to provide the time required to set up the longest shortest path in the given G.
- Δt_p: represents the time for which the prediction will be assumed to be a valid case. This should be defined carefully by the network operator so as to identify the true and false alarms after a certain time window (i.e. Δt_p).

The quality of the failure prediction is usually evaluated by two parameters: FP and FN; whereas, *Recall* and *Precision* are the two well-known metrics that are used to measure the overall performance.

$$Recall = \frac{TP}{TP + FN} \quad Precision = \frac{TP}{TP + FP} \tag{2}$$

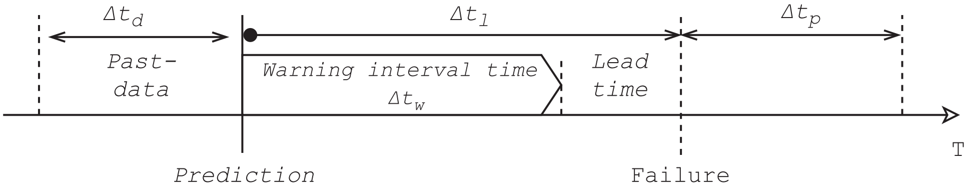


Fig. 2. Online failure prediction and time relations, Salfner et al. [44].

Table 2
Controller actions based on prediction.

Prediction	Action
TP	Select an alternative route
FP	Unnecessary/needless action
FN	Call the standard failure recovery

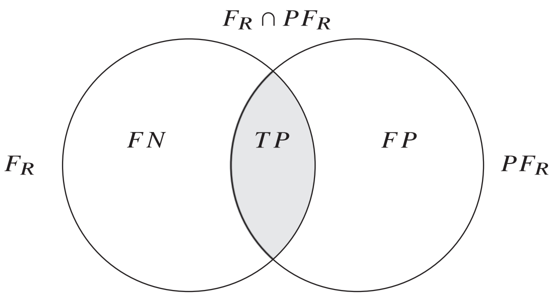


Fig. 3. Relation between prediction and failure sets.

Recall is defined as the ratio of the accurately captured failures to the total number of the certainly occurred failures, while, Precision is defined as the ratio of the correctly classified failures to the total number of the positive predictions. Correspondingly, SDN controller actions will now associate with the predicted and unpredicted situations as listed in Table 2.

On one hand, every false failure alarm will lead to an unnecessary reconfiguration for a particular set of routes in *Flow* and this will cause unwitting network instability. On the other hand, a controller needs to deal with the undetected failures in a similar way to the classical methods. Consequently, the more precise behaviour of prediction, the higher the percentage of network stability and service availability will be gained. Fig. 3 shows the relevance between the network model and the predictive model.

4.2. Failure event model

We have implemented an approach of generating failure events as it is very difficult to find a public network dataset that includes some useful details like failures, hence, we adopted an alternative approach by developing our failure model. This research intends to enhance the SDN fault tolerance and resilience through maximising the network service availability. Two basic metrics have been exploited in this model: *mean-time between failure* (*MTBF*) and *mean-time to recover* (*MTTR*); which are essential for calculating the availability and reliability of each network repairable component [5,50]. *MTBF* is defined as the average time in which a particular component functions before failing, calculated from: $\frac{\sum(start_{down\_time}-start_{up\_time})}{number\ of\ failures}$; while, *MTTR* is the average time required to repair a failed component. Each component, i.e., link, is characterized by its own values of both *MTBF* and *MTTR*, which are commonly independent from other components in the network. As a consequence of lacking real data, some metrics (such as cable length and *CC*) can be alternatively used for measuring the two availability metrics. According to [50], *MTBF* can be calculated as follows:

$$MTBF(hours) = \frac{CC \times 365 \times 24}{Cable\ Length} \qquad (3)$$

For instance, when *CC* is equal to 100 km, it means that per 100 km there will be on average one cut per year. Besides this, the *MTTR* of a link is influenced by its length [51], which expresses the fact that the longer link has a higher *MTTR* value. On this basis, we have designed the following formula for calculating the *MTTR* value for each link in the network.

$$MTTR(hours) = \gamma \times CableLength \qquad (4)$$

Where γ is defined as a parameter indicating the time required to fix the cable, which is measured by hour/kilometer format. Due to the fact that links are physically distributed in different locations and environments, therefore, γ differs from one link to another. In other words, even if some links have the same length, their γ could be different as it relies on the physical location and the ambient conditions. Further discussion is in Section 6.

5. Risk analysis

According to [52], risk can be defined as an attempt to answer the following three questions:

- What scenario could occur?
- What is the likelihood that scenario would occur?
- What is the consequence if the scenario does occur?

We consider these questions towards formulating the risk of failure in SDNs.

What scenario could occur? The *scenario* can be defined as any undesirable event such as failure. According to [53], there are three main types of failure scenarios that could affect the SDN networking system, these are: controller failure (including hardware and software), communication components failure (i.e. node and link) and application failure (e.g., bugs in application code). However, this paper considers the scenario of link failure only. Such scenario, breaks the service down when occurs. Therefore, finding alternative path is necessary. We define the set of link failure scenarios as F ranged over by variables $f_1, f_2, \ldots, f_n \in F$.

What is the likelihood that scenario would occur? The likelihood that a failure scenario disrupts the network services is conditional on the occurrence of the scenario. We address this question by the aid of online failure prediction that is in our case working based on a scenario's failure probability, p. Each failure scenario is associated with a p value that by nature ranges between 0 and 1, this will be further discussed in Section 6.

What is the consequence if the scenario does occur? We address this question by computing the percentage of loss or consequence, c, that might potentially happen when a failure scenario is predicted at an early stage. Each failure scenario might lead to some disconnections and service disruption. Therefore, the severity of adverse effects of each failure scenario varies. For instance, c_1 that was caused by f_1 might be different from c_2 that was caused by f_2, which would reflect the outage costs that would result from disrupting some of the network connections.

Table 3
List of failure scenarios.

Scenario	Probability	Consequence
f_1	p_1	c_1
f_2	p_2	c_2
.	.	.
.	.	.
.	.	.
f_n	p_n	c_n

Over a period of time, these questions would make a list of outcomes as exemplified in Table 3, where each ith row in the table can be represented as a triplet, i.e., $\langle f_i, p_i, c_i \rangle$.

Risk can be estimated by using such information as follows:

$$Risk = \{\langle f_i,\ p_i,\ c_i \rangle\}, \quad i = 1, 2, \ldots, n \tag{5}$$

Since we are considering the only link failure scenarios, $f_{(e_{ij})}$, we shall refine the definition of risk in (5). Accordingly, we redefine the risk to be the chance of damage that is determined by the combination of the probability of link failure and its consequence.

$$Risk_{f_{(e_{ij})}} = p_{(e_{ij})} \times c_{(e_{ij})} \tag{6}$$

To deduce the risk value, the two factors of (6), i.e., p and c, can be assessed independently. On one hand, the probability, p, depends on the efficacy of the online failure predictor at determining the likelihood of the incoming failure scenarios, which is, in this study, defined by a selective failure probability threshold value, T_{Ω}. On the other hand, for the consequence, c, it can be measured based upon the percentage of affected routes that would result from the anticipated scenario. In our case, we take the definition of such consequence one of the global network topological characteristics, namely *Edge Betweenness Centrality (EBC)* [47]. This is due to the fact that EBC is a direct indicator of the number of paths that would fail as a consequence of the failure of a particular link, therefore, providing a natural measure of risk consequence.

The EBC of a link e_{ij} is the total number of shortest paths between pairs of nodes that traverse the edge e_{ij} [47], which can be formulated as follows:

$$EBC_{e_{ij}} = \sum_{v_i \in V} \sum_{v_j \in V} \frac{\Gamma_{vi,vj} e_{ij}}{\Gamma_{vi,vj}} \tag{7}$$

where $\Gamma_{vi,\,vj}$ denotes the number of shortest paths between nodes vi and vj, while, $\Gamma_{vi,vj} e_{ij}$ denotes the number of shortest paths between nodes vi and vj and go through $e_{ij} \in E$. For instance, Fig. 4 demonstrates an example topology with an EBC value for each link in the network, which has been calculated based on Ulrik Brandes algorithm [48]. For instance, the EBC of e_{12} is calculated by the number of shortest paths containing the edge divided by all possible paths. Therefore, $\text{EBC}_{e_{12}} = 0.4$. This is because there are 20 possible paths in the example topology and 8 of them pass through the edge e_{12}. Given that network controller knows the demand traffic matrix between all pairs in the network, i.e., *Flow*. Therefore, Eq. (7) in our case is congruent with the following:

$$EBC_{e_{ij \in M}} = \frac{\Gamma_{flow} e_{ij}}{\Gamma_{flow}} \tag{8}$$

where Γ_{flow} denotes the total number of paths in *Flow* set, while, $\Gamma_{flow} e_{ij}$ denotes the number of paths in *Flow* set and pass through $e_{ij} \in M$.

With the above context in mind, the higher the EBC value of e_{ij}, which is a normalised value between 0 and 1, the more critical the link is and therefore, the higher the score indicating the consequences. This is because the outcome of failure for a link with high EBC will definitely lead to a huge number of path failures and therefore a higher percentage of negative impacts on the availability of network services.

Our goal in this analysis is to gauge the percentage of possible loss and provide such information to the concerned decision-making mechanism, i.e., the routing mechanism in our case. For more details about the existing risk analysis methods that fit SDNs, we refer the interested readers to [49].

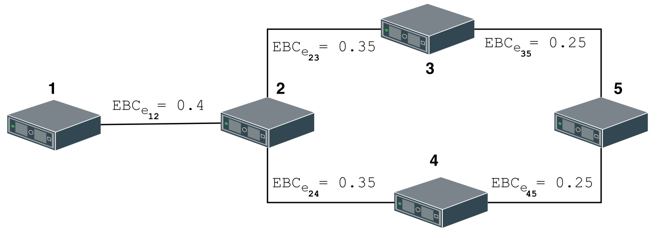


Fig. 4. Topology example with different EBC values.

6. The proposed framework

From a high level point of view, Fig. 5 illustrates the main components of our proposed framework where the *Smart Routing* (SR) and *Prediction Model* components are the primary contribution of our work. We discuss next in more detail the components we used and developed in this framework.

6.1. SDN controller

The proposed framework supports POX controller [54], which is an open source SDN controller written in python and it is more suitable for fast prototyping than other available controllers such as [55]. The standard OpenFlow protocol [3] is used for establishing the communication between the data and control planes, whereas the set of POX APIs can be used for developing various network control applications.

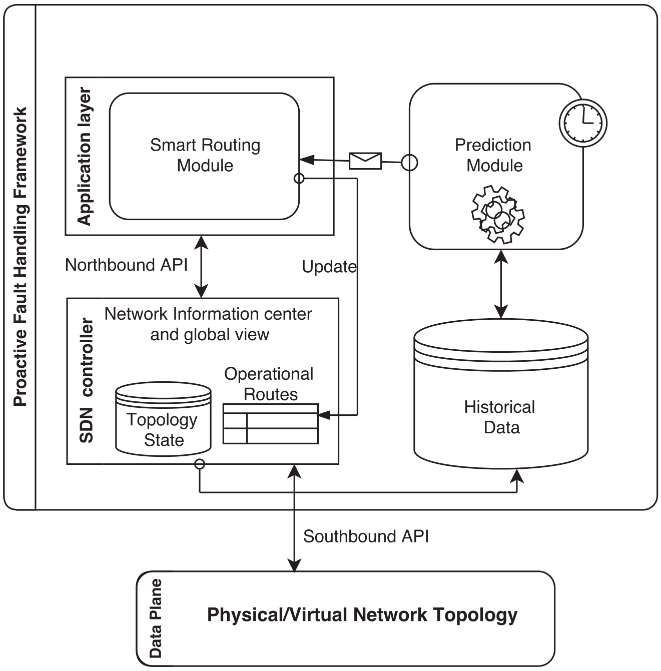


Fig. 5. Architecture of the proposed framework.

Table 4
Service availability and network flows relation.

Event	Flow	$src \rightarrow dst$ accessibility	T_C	Serviceability	Notes
–	$flow_x$	Yes	–	✔	Path is working
$flow_x \in F_R$	$flow_x$	No	T_D	✖	Path is not working
$flow_x \in F_R$	$flow_x$	No	T_{SP}	✖	Search for alternatives
$flow_x \in F_R$	$flow_x$	No	T_{Update}	✖	Path is restoring
–	$flow_x$	Yes	–	✔	Path is restored

6.2. Smart routing

Firstly, this module is responsible for maintaining and parsing the underlying network topology. Topology parameters such as the number of nodes and links, way of connection and port status can be detected via the Link Layer Discovery Protocol (LLDP) [56], which is one of the vital features of the current OpenFlow specification. The *openflow.discovery* [57], which is an already developed POX component that can be used to send specially crafted LLDP messages out of OpenFlow nodes so that the topological view over the data plane layer can be built.

This module will then convert the discovered network topology into a graph G representation for efficient management purposes. To do so, we utilised the Networkx tool [58], which is a pure python package with a set of powerful functions for manipulating network graphs. When the network starts working and after shaping the data plane topology, the shortest path for each $flow \in Flow$ is configured by the appointed SP_X algorithm, which thereafter is stored in the *Operational Routes* table that is specified to contain all the desired working (healthy) paths.

In order to perceive how the link failure incident could affect the configured paths from the perspective of service availability and convergence time, we provide a simple example in Table 4 in which the service deterioration of the $flow_x$ due to link failure incident is highlighted.

To maintain the *Operational Routes* table, two algorithms have been implemented each with its own view in respect to keep the *Flow* maintained. Algorithm 1 depicts the baseline shortest path routing strategy (henceforth called Baseline Routing (BR)), which is currently performed by the SDN controller. We specify Dijkstra's [40] algorithm, with complexity $O(|V| + |E|\ log\ |V|)$, as the shortest path finder approach for Algorithm 1, which we denote by SP_D instead of SP_X. So, the SP_D is a Dijkstra function that can be applied on any $flow_{set}$ to return only one unique shortest path.

When the OpenFlow controller reports a link failure event, every path suffering from that failure will be detected and then two operations will be issued by the controller. First, a *Remove*, denoted by OF_{Remove}, command is sent to all the routers that belong to each failed path in *Flow* as a step to weeding out the incorrectly working entries, then an alternative route will be computed for every affected *flow*. The new flow entries of the alternative path are then forwarded to the relevant routers of each *flow* through the *Install*, denoted by $OF_{Install}$, command. Each modified *flow*, i.e., assigned to alternative, will be stored in a special set that is called *the Labeled Flow* (*LF*), where: $LF \subset Flow$ and with length of n. This is to indicate that each $flow \in LF$ is in a sub-optimal state. The recovery from link failure procedure is demonstrated in line (1–13). However, the algorithm also includes the reversion process after a failure is reformed (line 15–32), which is no less important than the recovery process [59], and also to take into account the percentage of routing flaps that is necessary for later analysis. In fact, we developed this algorithm for comparison purposes only against Algorithm 2 . Therefore, it does not reflect a contribution of this paper.

In contrast (and unlike Algorithm 1), Algorithm 2 is one of the main contributions of this work that exploited the prediction information towards enhancing the service availability and the fault tolerance of SDNs. This algorithm depends on Bhandari's algorithm for finding K edge-disjoint paths [60], which has been utilised as a complementary to build the smart routing strategy. We denoted Bhandari's algorithm as SP_B in place of SP_X.

Algorithm 1: Baseline routing (BR).

```
   On Normal: $\forall\ flow \in Flow$ : Set Primary Path as flow. $flow \in SP_D(flow_{set})$
   On Failure : Do the following procedure
1  if Link failure reported then
2      foreach $e_{ij} \in F$ do
3          Compute: $F_R$
4      end
5      do
6          $OF_{Remove}\ (flow)$
7          $flow_{set} := flow_{set} - \{flow\}$
8          $flow := SP_D(flow_{set})$
9          $OF_{Install}\ (flow)$
10         $LF \leftarrow flow$
11         $F_R := F_R - \{flow\}$
12     while $F_R \neq \emptyset$;
13 end
14 c := 0
15 if Link repair reported then
16     do
17         if $flow_c$ is currently optimal then
18             Do nothing
19             c := c + 1
20         end
21         if $flow_c$ is currently sub-optimal then
22             $OF_{Remove}\ (flow_c)$
23             $flow_c := SP_D(flow_{c_{set}})$
24             $OF_{Install}\ (flow_c)$
25             $LF := LF - \{flow_c\}$
26             c := c + 1
27         end
28         if number of links = $E_{len}$ then
29             $LF := empty$
30         end
31     while $c \leqslant LF_{len}$;
32 end
```

Thereon, we consider the SP_B as a function that is specified to compute 2 link-disjoint paths with the least total cost for any given pair of nodes (i.e. *src* and *dst*) or $flow_{set}$. For the purpose of distinguishing between the two returned paths of SP_B, we denote the first path as $flow_{b_1}$ and the second disjoint one as $flow_{b_2}$. The time complexity of SP_B is different from the SP_D, which is a polynomial that is equivalent to $O((K+1).|E| + |V|\ log\ |V|)$. The pseudo code of smart routing is demonstrated in Algorithm 2, in which the $flow_{b_1}$ is initially selected to represent the primary path for each *flow* in the network.

The network controller will then start listening to the prediction module, which will be discussed in the next section, for the

Algorithm 2: Smart routing (SR).

```
    Input  : Network topology $G(V, E)$, $M$
    Output: $PF_R \approx \emptyset$
 1  $\forall$ $flow \in Flow$ : Set Primary Path as $flow_{b_1}$ . $flow_{b_1} \in SP_B(flow_{set})$
 2  if $M = \{m\}$ then
 3      $PF_L \leftarrow \bar{e}_{ij}$
 4  end
 5  foreach $\bar{e}_{ij} \in PF_L$ do
 6      Compute: $PF_R$
 7  end
 8  $EBC_{\bar{e}_{ij}} = \frac{PF_{R_{len}}}{Flow_{len}}$
 9  $Risk_{\bar{e}_{ij}} = p(\bar{e}_{ij}) \times EBC_{\bar{e}_{ij}}$
10  if $Risk_{\bar{e}_{ij}} \geqslant Risk_{T_\omega}$ then
11      do
12          $OF_{Install}$ $(flow_{b_2}$ . $flow_{b_2} \in SP_B(flow_{set}))$
13          $OF_{Remove}$ $(flow_{b_1}$ . $flow_{b_1} \in SP_B(flow_{set}))$
14      while $PF_R \neq \emptyset$;
15      Wait: $\Delta t_p$
16      if $\bar{e}_{ij} \in F$ then
17          Mark as: $TP$
18          $LF \leftarrow PF_R$
19      else
20          Mark as: $FP$
21          do
22              $OF_{Install}$ $(flow_{b_1}$ . $flow_{b_1} \in SP_B(flow_{set}))$
23              $OF_{Remove}$ $(flow_{b_2}$ . $flow_{b_2} \in SP_B(flow_{set}))$
24          while $PF_R \neq \emptyset$;
25      end
26  end
27  $PF_R = \emptyset$
28  if $[\ F = (e_{ij}) \wedge (e_{ij} \notin M)\ ] \vee [\ F = (e_{ij}) \wedge (e_{ij} \in M) \wedge (Risk_{\bar{e}_{ij}} < Risk_{T_\omega})\ ]$ then
29      Mark as: $FN$
30      Call Algorithm1
31  end
32  if Link repair reported then
33      Call Algorithm1
34  end
```

potential of future incidents. When a new message (m) is received, the controller will firstly construct the potential failed list, which contains the information about link which is expected to fail in the near future as described in (line 2–4). Secondly, the route (or routes) which might be affected according to the predicted failure message will be computed as a preparatory step to replace them (line 5–7). After identifying the routes that may possibly fail, the *EBC* for the predicted link will be calculated as a step towards measuring the risk (line 8–10). If the risk value is below the risk threshold, then the prediction information will be ignored and no action will be taken. Otherwise, the flow entries of the newly computed disjoint path from the second step will be installed through using the *Install* command. This is done by adjusting the disjoint path rules with lower priority than the primary path to avoid the conflict of matching and action process. Following this step, the forwarding rules of the risky primary paths will need to be deleted in order to use TCAM resources efficiently.

This needs to be done in a similar procedure to the installation but with the *Remove* command as demonstrated in (line 11–14). After swapping the primary path due to an expected failure, this action will be considered as the correct decision for a certain period of time (i.e. Δt_p) as indicated in line 15. To examine the substantiality of the changing routes decision, the link that was anticipated to go down within Δt_l will be compared against the failure set *F*. On one hand, and if the link exists, then the prediction will be marked as *TP*, and each $flow \in PF_R$ will be labeled as suboptimal, in addition, the reconfigured paths will store in *LF* (line 16–18). On the other hand, the prediction will be considered as *FP* and in such a case it is necessary to reset the primary path to its initial state (i.e. optimal) as deliberated in (line 19-25). In case when there is a failure that was not captured by the prediction module, such a case is considered as *FN* and the failure in such situations is tackled by calling Algorithm 1 as outlined in (line 28–30). Finally, Algorithm 1 will also be invoked when a failed link is repaired (line 32–34).

6.3. Prediction module

In this work, this module is placed on top of the parsed network topology state that gained from the network controller as a result of lacking historical data. We consider each link as an independent object of link class. The link class contains a set of attributes, which currently includes eight attributes as shown in Fig. 6. The link attributes are used to control the up and down events. In the current implementation, we used the priority queue, *Q*, as a pool to hold all the non-faulty links. On one hand, Eqs. (3) and (4) are essential for computing the two static attributes (*MTBF* and *MTTR*) of each link. For (3), we rely on the topologies information in Section 7.3 and by assuming that *CC* equals the minimum cable length in a network. While, for (4) we used the uniform distribution to generate γ for each link independently. On the other hand, the six remaining ones are described as follows:

- ID : a numerical unique value (i.e. $1, 2, \ldots, n$) assigned to the link to represent the link identification number.
- F_Count : a counter to contain the number of times the link has failed.
- Length : represents the link's length in km, which is derived from the topology specification.
- Next_F : refers to the *next time to failure* of link, which controls the process of moving the link into and out-of the *Q*. In other words, this attribute determines the link life span in the *Q* where the link will be dequeued when its *Next_F* equals to zero.
- Probability_F : registers the current failure probability, p, of the link. For instance, the *Probability_F* of the link (j) is arrived through: $\frac{F\_Count(ID_j)}{\sum_{i=1}^{n} F\_Count(ID_i)} \times 100$, where n is the length of *Q*.
- Status : reflects the current state of the link as either operational or faulty.

On this basis, we have placed our online predictor scheme, i.e., represented by Algorithm 3 , on top of the priority queue in order to send encapsulated messages about the links which satisfy the following two conditions (as described in line 2–9):

✓ The probability of failure is greater than or equal to the threshold T_Ω.
✓ The leading time (i.e. Δt_l) is less than or equal to the *next time to failure.*

Failure Decision (FD) is a Boolean function that randomly generates True and False values for each link that satisfies the threshold condition, i.e., T_Ω. When FD is True, a failure event is generated by putting the current link, i.e., $F_{(Q_{ptr})}$, down if Δt_l is satisfied. Otherwise, when FD is False then no failure event will be generated. Algorithm 3 is used only for evaluation purposes so that True and

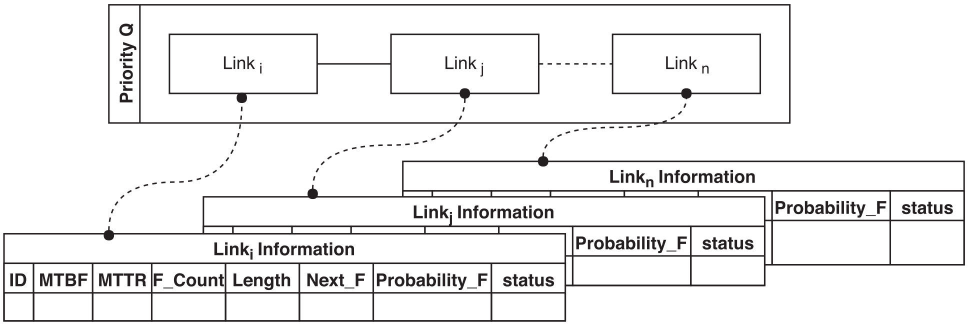


Fig. 6. Representation of links in priority queue.

Algorithm 3: Alarm message generator (M).

```
Input  : $G(V, E)$
Output: $M$
1  while $(Q! = \emptyset)$ do
2      if $Probability\_F_{(Q_{ptr})} \geqslant T_{\Omega}$ then
3          if $FD$ = $True$ then
4              Go To: 8
5          else
6              Go To: 15
7          end
8          Compute: $\Delta t_l$
9          if $Next\_F_{(Q_{ptr})} \geqslant \Delta t_l$ then
10             Wait: $Next\_F_{(Q_{ptr})} - \Delta t_l$
11             Generate: $(m, \bar{e}_{ij_{(Q_{ptr})}})$
12         else
13             $\Delta t_l$ is not satisfied
14         end
15     end
16     Wait: $Next\_F_{(Q_{ptr})} = 0$
17 end
```

False alarms can be made. Hence, the actual link failure prediction method is outside the scope of this paper.

7. Experimental design and implementation

Since smart routing is aimed to enhance the SDN fault tolerance in the context of network service availability, we have implemented some metrics for fair comparison between the traditional SDN and the proposed system. We also show in this section the adopted network topologies that have been utilised in our experiments.

7.1. Availability measurements

Considering the convergence time that is required to shift from a failed or non-operational path to an alternative or backup one, which conforms with Eq. (1). This convergence process is definitely drive to some damages in the network availability and causing service unavailability. This issue results from the unavailability of the affected path to the service for a certain amount of time, as demonstrated in Table 4. In order to identify the serviceable, which are denoted by "Yes", and unserviceable, which are denoted by "No", *flows* with respect to some failure events, we formulated this problem as follows:

$$(flow \cap Q) = flow \Rightarrow Yes$$

$$(flow \cap Q) \subset flow \Rightarrow No$$

where, "Yes" and "No" can be obtained by intersecting each *flow* ∈ *Flow* against the *Q*. The *flow* is subjected to "Yes" when all its forming edges reside in the *Q*, otherwise, the *flow* will be considered as unserviceable and subjected to "No". By knowing the number of serviceable and unserviceable *flows*, the service unavailability and thus the service availability can be measured.

The service unavailability of SDN (U_{SDN}) over a given interval time with a certain number of failure events, which are denoted by *ev*, can be arrived through the following:

$$U_{SDN}(Flow, G) = \frac{\sum_{i=1}^{ev} flow \in FlowNo}{ev \ \times \ Flow_{len}} \tag{9}$$

Whereas, for smart routing it is important to further consider the impact of *Recall* values. Hence, the service unavailability of *SR* (U_{SR}) can be arrived by the following:

$$U_{SR}(Flow, G) = (1 - Recall) \times (U_{SDN}(Flow, G)) \tag{10}$$

Consequently, the availability A_x, with $x = \ SDN \ or \ SR$, can be arrived through the following:

$$A_x = 1 - U_x \tag{11}$$

7.2. Routing instability measurements

In traditional networks, routing protocols (e.g. IGP [61]) perform two routing changes as a reaction to every single failure, one time when a failure occurs and another when a failure is repaired. In fact, both changes are essential for the QoS where the first change is for the purpose of service availability, while, the goal of the second one is to return back from the backup (i.e. sub-optimal) to the primary (i.e. optimal) path again. In contrast, SDN architecture brings centralisation and programmability to the scene, therefore, traditional distributed protocols are independent of the SDN architecture. Maintaining the optimal path (e.g. minimum hops in our case) of each *flow* will require a continuously adaptive strategy that will be responsible for replacing each sub-optimal *flow* with the optimal one after it becomes serviceable. To do so, we assume that each alternative *flow* is additionally stored in *LF* as mentioned in Section 6.

For an SDN, the routing flaps (denoted by *RF*) can be measured by means of link *up* (denoted by u_f) and *down* (denoted by d_f) as follows:

$$RF_{SDN} = \sum_{flow \in LF} u_f + \sum_{flow \in F_R} d_f \tag{12}$$

On one hand, and according to (12), after each link down event; a new route for each *flow* ∈ F_R is required, which then leads to a first routing change for each *flow*. On the other hand, and after each link up announcement, the controller will need to check the

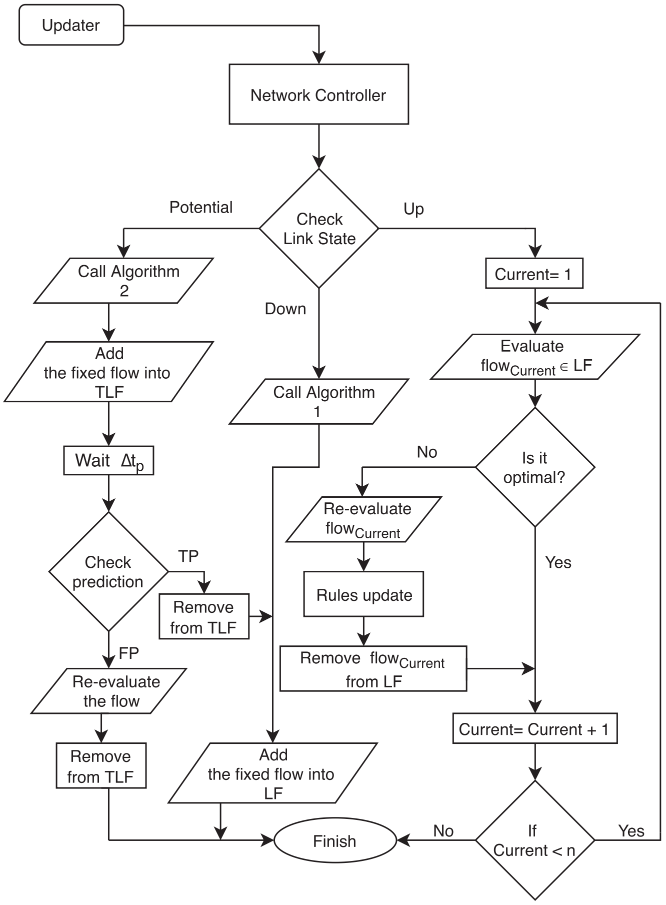


Fig. 7. Flow chart of routing flaps.

state of each labeled *flow* in *LF* to determine if itâs still the optimal choice. If so, then no change will be made, otherwise, rerouting is required and therefore it will result in another routing change.

However, for the smart routing mechanism, it is necessary to consider the three prediction parameters also (i.e. *FN*, *TP* and *FP*), as follows:

$$RF_{SR} = \sum_{flow\in F_R} FN_f + \sum_{flow\in PF_R} TP_f + \sum_{flow\in PF_R} FP_f + \sum_{flow\in LF} u_f \qquad (13)$$

According to (13), the FN_f is equivalent to d_f in (12) as it reflects the actual failure events that have not been captured by the prediction module, while the remaining are as follows:

- Each true prediction will lead to a first reroute flap that gives the advantage of avoiding an upcoming failure event. While, the second flap will be similar to the scenario of RF_{SDN} through inserting the *flow* into the *LF* and the next flap builds upon the link restoration u_f.
- Each false prediction leads into two useless flaps, one when the prediction triggers an alarm, in such a case each potential *flow* will be added to the *Temporary Labeled Flow* set (*TLF*), as a transient step before it recognises the prediction was false. The second flap will perform when Δt_p expires.

We provide a deep overview of the process of measuring the number of routing flaps in the flow chart of Fig. 7, which also shows how the *LF* is adjusted in the scenario of the two algorithms, i.e., Algorithms 1 and 2. Since all actions are associated with the link state, in this work, we utilise the OpenFlow protocol to reflect the data plane links changing state. This is by relying on the *Loss of Signal* (LoS) that detects link failures by depending

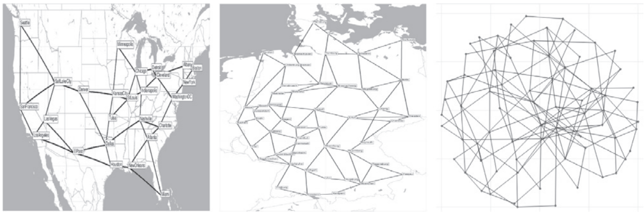

(a) janos-us (b) germany50 (c) waxman

Fig. 8. Experimental topologies.

Table 5
Topologies' characteristics.

Topology	Nodes	Edges	Min$_{len(e_{ij})}$	Max$_{len(e_{ij})}$
janos-us	26	42	145 km	1127 km
germany50	50	88	36 km	236 km
waxman	70	140	15 km	1099 km

on OpenFlow `PORT-STATUS` messages. In addition, the proposed prediction module produces further information about the potential failures. Both LoS and prediction information will be delivered to the network controller through the *Updater* in order to apply the appropriate action as illustrated in the above flow chart.

7.3. Simulated network topologies

In order to evaluate the proposed method, we have modelled 3 network topologies as depicted in Fig. 8. Both, (a) janos-us and (b) germany50 represent real network topology instances that defined in [62]. However, (c) waxman is a synthetic topology that is created by the Internet topology generator Brite [63] through using the well-known Waxman model [64]. Waxman's model is a geographical approach that connects the distributed routers in a plane on the basis of the distance among them, which is given by the following formula:

$$\mathbb{P}(\{v_i, v_j\}) = \beta\ exp^{\frac{-d(v_i,v_j)}{L\alpha}} \qquad (14)$$

where $0 < \alpha$ and $\beta \leq 1$. d represents the distance between v_i and v_j, while L represents the maximum distance between any two given nodes. The number of links among the generated nodes is associated with the value of α in a directly proportional manner, while the edge distance increases when the value of β is incremented. We used Brite to generate a large-scale network topology in comparison to the others (e.g. when the number of edges or nodes $\geq$ 100). The characteristics of all the modelled topologies are detailed in Table 5.

7.4. Implementation

In order to validate our approach, the proposed framework is built on top of the POX controller. The implementation code of the current framework is made available on the Github platform [65]. The proposed framework is evaluated by using the container-based emulator, Mininet [66]. Mininet is a widely used emulation system, as evidenced in a recent survey [4], for evaluating and prototyping SDN protocols and applications. It can also be used to create realistic virtual networks, running real kernel, switch and application code, on a single machine (VM, cloud or native). Our experiments were designed based on the topologies that we illustrated in the preceding section. Since one of our experimental topologies was designed via Brite, we utilised the Fast Network Simulation Setup (FNSS) [67]. FNSS is a python-based toolchain simulator that can be used to facilitate the process of network experiments. It provides a wide range of functions and adapters that

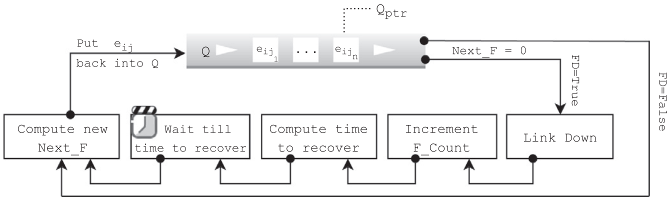


Fig. 9. Flow diagram of a link's life cycle in the Queue.

allow network researchers to parse graphs from different topology generators (e.g. Brite) in order to be compatible with and/or to interface with other simulator/emulator tools, such as Mininet. Based on the *failure event model* (Section 4.2), the general reliability theory [68] has been utilised to generate failure events using the exponential distribution $(mean = MTBF)$ for the next time to failure of each link, and lognormal distribution $E(\mu, \sigma)$ with $\mu = \log(MTTR) - ((0.5) \times \log(1 + ((0.6 \times MTTR)^2/MTTR^2)))$ and $\sigma = \sqrt{\log(1 + ((0.6 \times MTTR)^2/MTTR^2}$ for time to recover. Regarding, failure anticipation, false and true positive have been generated during the simulated time using the uniform distribution following the specified threshold value. Fig. 9 summarises the simulated link queuing system that is correlated to the two metrics of reliability, i.e., MTBF and MTTR. In order to dispatch the prediction information that is necessarily important to the SR module, the distributed messages framework (ZeroMQ [69]) was exploited to carry the alarm messages, *M*, from the prediction module to the network controller interface. In some network *flow* conditions it will activate the SR module to begin a possible reconfiguration. In the emulation environment, we employed two servers; one acts as the OpenFlow controller and the other to simulate the network topologies. For each server, we used Ubuntu v.14.04 LTS with Intel Core-i5 CPU and 8 GB RAM.

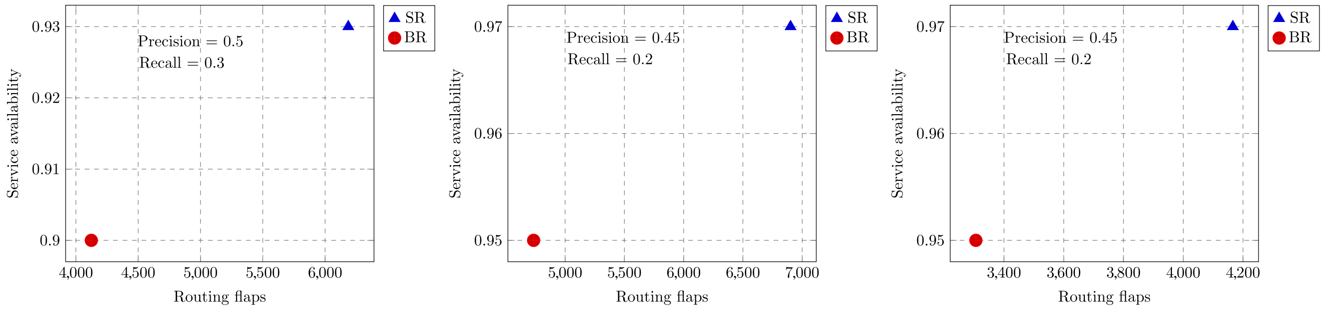
8. Comparison and key advantages of SR

In this section, we present comparison and evaluation of the proposed method versus the default SDN technique (i.e. BR). To do so, the study has been conducted on the three topologies that were summarised in Table 5. To simulate the three topologies, we ran the emulator for 144 h, i.e., each experimental topology was simulated in the system for 48 h. Fig. 10 shows the obtained results from the three topologies based on parameter settings of $T_\Omega = 0.25$, $T_\omega = 0.1$, $\Delta t_l = 120$ s and $\Delta t_p = 30$ s. Keep in mind, and as discussed earlier, the T_Ω and T_ω values can be selected by the network operator or by using additional algorithms (i.e. machine learning) to identify the near optimal values. Since the main goal of smart routing is to enhance the network service availability, we plot for each network that which gives the BR and SR mechanisms for the service availability percentage (Y-axis) and the rate of routing flaps (X-axis). Furthermore, for SR, the performance of the online failure predictor represented by the values of *Recall* and *Precision* are considered and reported respectively to each topology. In fact, *Recall* value has a crucial impact on the service availability in the SR scheme, however, *Precision* value has an impact on the unnecessary routing changes.

On one hand, it can be clearly observed that SR outperformed the BR in providing network service availability for all test cases. In spite of the low *Recall* values (i.e. 0.2-0.3), there is still a gain in service availability. It can also be observed that *janos-us* topology gained the highest improvement percentage in the service availability and this is because its *Recall* value is greater than that of the other topologies.

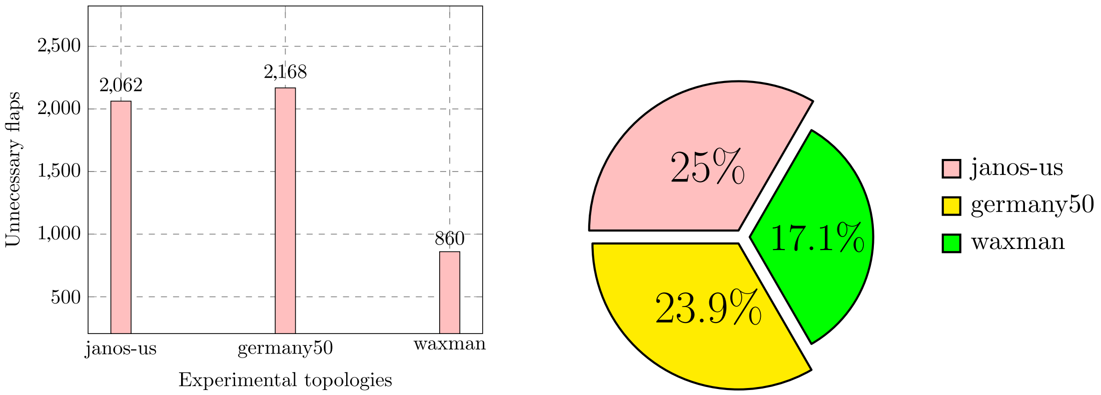
On the other hand, the rate of the routing flaps generated by SR is always higher than the BR. This disadvantage comes as a trade-off for improving the network service availability. Given that the routing instability by means of unnecessary flaps is correlated with the value *Precision*, we have measured the only useless flaps that were generated during the simulation time and for each topology as shown in Fig. 11. Fig. 11(a) shows the only unnecessary routing changes that have been reported based on the *FP* rate of each topology, where each single *FP* is associated with two useless flaps, that is, one for the reconfiguration and the other for the reversion. However, Fig. 11(b) shows the percentage of useless routing flaps for each topology in comparison with the total number of flaps. In the worst case scenario the routing flaps did not exceed 25%. Although *janos-us* topology has the highest *Precision* value, it yielded a relatively high percentage of useless flaps and this is because the number of links in the topology is low, hence, it is highly likely that each single link is associated with a large number of routes in contrast to the other two topologies. It is also clearly evident that the online failure prediction plays a significant role in both service availability (by *TP*) and routing flaps (by *FP*). Based upon the experiments and simulations, we have some observations, as follows:

- Some alternative routes are considered as optimal after receiving an update message, even though the received update is not involved in its conforming path. The reason is that the current system defines the optimal path based on hops number. Therefore, each alternative path that has the same number of hops as the optimal one will be considered to be an optimal path. It might not be the case if the adopted routing constraint is not the number of hops, i.e., using a specified cost function with different parameters such as bandwidth, congestion, energy, etc.
- In some cases it is barely able to find two-disjoint paths and therefore, sometimes if a path faces two successive failure alarms on its forming links, then no change will be made. Hence, we used (≈) instead of (=) in the output of



(a) janos-us (b) germany50 (c) waxman

Fig. 10. Routing flaps and service availability.



(a) Useless flaps rate

(b) Useless flaps percentage

Fig. 11. Routing instability measurements.

Algorithm 2, to imply that an entirely empty PF_R cannot be always guaranteed.

- It is also possible that each $flow \in LF$ may face one or more risky links, thus in such a case the entangled *flow* state will be the same (i.e. sub-optimal).
- In some cases and when the $Next\_F < 2\ min$, the controller will ignore the prediction if it is generated, as in such cases the Δt_l is not satisfied and therefore the controller will not have enough time for the reconfiguration.

9. Conclusion and future directions

This paper has demonstrated the promise of using online failure prediction to enhance the SDN service availability. Since the network service availability is a well established research area, its implications for OpenFlow networks are limited. We presented Smart Routing to tackle the problem of data plane link failures in SDNs. The proposed approach differs from the existing contributions by allowing the SDN controller to have a time window in order to reconfigure the network before the anticipated link failure takes place. With such approach, the interruption of the network services caused by link failures can be reduced and therefore it brings significant benefits to the network service availability. We demonstrated how the proposed model can be implemented using a couple of new algorithms that extract the risky links from the currently-used paths and hence none of these paths will be affected when a risky link fails. The performance of the proposed approach is tested and evaluated through extensive simulation experiments on various real and synthetic network topologies conducted with the link failure event model. The experimental findings show clearly the effectiveness of the proposed method in enhancing the SDN service availability. Unfortunately, the flaps rate that can be resulted from the failure prediction may lead to network instability, especially when it reaches a high rate. For this purpose, we measured the percentage of the unnecessary routing flaps and in the worst case scenario, the rate was 25%, which is nearly reasonable in practice.

As future work, we will position the study in the setting of machine learning and signal processing towards achieving that the decision will be made according to the optimal threshold value of the probability of failure. We are also planning to extend this study to consider some disaster situations where drastic failure scenarios can lead to multiple link failures with high network availability degradation and packet loss rates. In such scenarios, one needs to consider different, possibly less predictable, metrics of failure. Additionally, we plan to consider more complex scenarios where we consider not only link failures but also other forms of failure, e.g. controller, node and application failures.

Declaration of Competing Interest

We have no conflicts of interest to declare.

CRediT authorship contribution statement

Ali Malik: Conceptualization, Data curation, Investigation, Methodology, Project administration, Software, Validation, Writing - original draft. **Benjamin Aziz:** Conceptualization, Supervision, Writing - review & editing, Investigation, Validation. **Mo Adda:** Supervision, Data curation, Writing - review & editing, Investigation, Validation. **Chih-Heng Ke:** Data curation, Investigation, Validation, Software, Writing - review & editing.

Acknowledgements

The authors thank the anonymous reviewers for their valuable and thoughtful comments.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2020.107104.

References

[1] P. Lin, J. Bi, H. Hu, T. Feng, X. Jiang, A quick survey on selected approaches for preparing programmable networks, in: Proceedings of the 7th Asian Internet Engineering Conference, ACM, 2011, pp. 160–163.

[2] N. Feamster, J. Rexford, E. Zegura, The road to SDN: an intellectual history of programmable networks, ACM SIGCOMM Comput. Commun. Rev. 44 (2) (2014) 87–98.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.

[4] D. Kreutz, F.M. Ramos, P.E. Verissimo, C.E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: a comprehensive survey, Proc. IEEE 103 (1) (2015) 14–76.

[5] J.C. Laprie, Dependability: basic concepts and terminology, in: Dependability: Basic Concepts and Terminology, Springer, Vienna, 1992, pp. 3–245.

[6] J. Ai, Z. Guo, H. Chen, G. Cheng, Improving the routing security in software-defined networks, IEEE Commun. Lett. 23 (5) (2019) 838–841.

[7] T. Wang, Z. Guo, H. Chen, W. Liu, BWManager: mitigating denial of service attacks in software-defined networks through bandwidth prediction, IEEE Trans. Netw. Serv. Manag. 15 (4) (2018) 1235–1248.

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

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

[10] G. Iannaccone, C.N. Chuah, R. Mortier, S. Bhattacharyya, C. Diot, Analysis of link failures in an IP backbone, Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement (pp. 237–242). ACM.

[11] F. da Rocha, C. Paulo, E.S. Mota, A survey on fault management in software-defined networks, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2284–2321.

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

[13] Z. Guo, W. Feng, S. Liu, W. Jiang, Y. Xu, Z.L. Zhang, Retroflow: maintaining control resiliency and flow programmability for software-defined WANs, 2019, arXiv:1905.03945.

[14] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.N. Chuah, C. Diot, Characterization of failures in an IP backbone, in: INFOCOM 2004. Twenty-Third AnnualJoint Conference of the IEEE Computer and Communications Societies, volume 4, IEEE, 2004, March, pp. 2307–2317.

[15] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.N. Chuah, Y. Ganjali, C. Diot, Characterization of failures in an operational IP backbone network, IEEE/ACM Trans. Netw. 16 (4) (2008) 749–762.

[16] I.F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A roadmap for traffic engineering in SDN-OpenFlow networks, Comput. Netw. 71 (2014) 1–30.

[17] J.P. Vasseur, M. Pickavet, P. Demeester, Network recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS, Elsevier, 2004.

[18] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takcs, P. Skldstrm, Scalable fault management for OpenFlow, in: Communications (ICC), 2012 IEEE International Conference, IEEE, 2012, pp. 6606–6610.

[19] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, P. Castoldi, OpenFlow-based segment protection in ethernet networks, J. Opt. Commun. Netw. 5 (9) (2013) 1066–1075.

[20] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, H.J. Chao, STAR: preventing flow-table overflow in software-defined networks, Comput. Netw. 125 (2017) 15–25.

[21] Z. Guo, Y. Xu, R. Liu, A. Gushchin, K.Y. Chen, A. Walid, H.J. Chao, Balancing flow table occupancy and link utilization in software-defined networks, Future Gener. Comput. Syst. 89 (2018) 213–223.

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

[23] D. Staessens, S. Sharma, D. Colle, M. Pickavet, P. Demeester, Software defined networking: Meeting carrier grade requirements, in: Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on, IEEE, 2011, pp. 1–6.

[24] S. Sharma, D. Staessens, D. Colle, M. Pickavet, P. Demeester, OpenFlow: meeting carrier-grade recovery requirements, Comput. Commun. 36 (6) (2013) 656–665.

[25] H. Kim, M. Schlansker, J.R. Santos, J. Tourrilhes, Y. Turner, N. Feamster, Coronet: fault tolerance for software defined networks, in: Network Protocols (ICNP), 2012 20th IEEE International Conference on, IEEE, 2012, pp. 1–2.

[26] M. Luo, Y. Zeng, J. Li, W. Chou, An adaptive multi-path computation framework for centrally controlled networks, Comput. Netw. 83 (2015) 30–44.

[27] Y. Jinyao, Z. Hailong, S. Qianjun, L. Bo, G. Xiao, HiQoS: an SDN-based multipath QoS solution, China Commun. 12 (5) (2015) 123–133.

[28] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A.W. Moore, OFLOPS: an open framework for OpenFlow switch evaluation, in: International Conference on Passive and Active Network Measurement, Springer, Berlin Heidelberg, 2012, pp. 85–95.

[29] X. Jin, H.H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, R. Wattenhofer, Dynamic scheduling of network updates, in: ACM SIGCOMM Computer Communication Review (Vol. 44, No. 4, ACM, 2014, pp. 539–550.

[30] G. Bianchi, M. Bonola, A. Capone, C. Cascone, OpenState: programming platform-independent stateful OpenFlow applications inside the switch, ACM SIGCOMM Comput. Commun. Rev. 44 (2) (2014) 44–51.

[31] A. Capone, C. Cascone, A.Q. Nguyen, B. Sanso, Detour planning for fast and reliable failure recovery in SDN with OpenState, in: Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the, IEEE, 2015, pp. 25–32.

[32] C. Cascone, L. Pollini, D. Sanvito, A. Capone, B. Sanso, SPIDER: fault resilient SDN pipeline with recovery delay guarantees, in: NetSoft Conference and Workshops (NetSoft), 2016 IEEE, IEEE, 2016, pp. 296–302.

[33] S.A. Astaneh, S.S. Heydari, Optimization of SDN flow operations in multi-failure restoration scenarios, IEEE Trans. Netw. Serv. Manag. 13 (3) (2016) 421–432.

[34] A. Malik, B. Aziz, M. Adda, C.H. Ke, Optimisation methods for fast restoration of software-defined networks, IEEE Access 5 (2017) 16111–16123.

[35] A. Malik, B. Aziz, C.H. Ke, H. Liu, M. Adda, Virtual topology partitioning towards an efficient failure recovery of software defined networks, in: Machine Learning and Cybernetics (ICMLC), 2017 International Conference on, IEEE, pp. 646–651.

[36] A. Malik, B. Aziz, A. Al-Haj, M. Adda, Software-defined networks: a walkthrough guide from occurrence to data plane fault tolerance (No. e27624v1). PeerJ preprints, 2019.

[37] S.S. Lee, K.Y. Li, K.Y. Chan, G.H. Lai, Y.C. Chung, Software-based fast failure recovery for resilient OpenFlow networks, in: Reliable Networks Design and Modeling (RNDM), 2015 7th International Workshop on, IEEE, 2015, pp. 194–200.

[38] M. Desai, T. Nandagopal, Coping with link failures in centralized control plane architectures, in: Communication Systems and Networks (COMSNETS), 2010 Second International Conference on, IEEE, 2010, pp. 1–10.

[39] S.S. Lee, K.Y. Li, K.Y. Chan, G.H. Lai, Y.C. Chung, Path layout planning and software based fast failure detection in survivable OpenFlow networks, in: Design of Reliable Communication Networks (DRCN), 2014 10th International Conference on the, IEEE, 2014, pp. 1–8.

[40] E.W. Dijkstra, W. E., A note on two problems in connexion with graphs, Numer. Math. 1 (1) (1959) 269–271.

[41] B. Vidalenc, L. Ciavaglia, L. Noirie, E. Renault, Dynamic risk-aware routing for OSPF networks, in: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, IEEE, 2013, pp. 226–234.

[42] A. Medem, R. Teixeira, N. Feamster, M. Meulle, Joint analysis of network incidents and intradomain routing changes, in: Network and Service Management (CNSM), 2010 International Conference on, IEEE, 2010, pp. 198–205.

[43] C. Labovitz, G.R. Malan, F. Jahanian, Origins of internet routing instability, in: INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, IEEE, 1999, pp. 218–226.

[44] F. Salfner, M. Lenk, M. Malek, A survey of online failure prediction methods, ACM Comput. Surv. (CSUR) 42 (3) (2010) 10.

[45] A. Medem, R. Teixeira, N. Usunier, Predicting critical intradomain routing events, in: Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE, IEEE, 2010, pp. 1–5.

[46] R.S. Mangoubi, Robust Estimation and Failure Detection: A Concise Treatment, Springer Science & Business Media, 2012.

[47] L. Lu, M. Zhang, Edge betweenness centrality, in: Encyclopedia of Systems Biology, Springer, New York, NY., 2013, pp. 647–648.

[48] U. Brandes, On variants of shortest–path betweenness centrality and their generic computation, Soc. Netw. 30 (2) (2008) 136–145.

[49] S. Szwaczyk, K. Wrona, M. Amanowicz, Applicability of risk analysis methods to risk-aware routing in software-defined networks, in: 2018 International Conference on Military Communications and Information Systems (ICMCIS), IEEE, 2018, pp. 1–7.

[50] S. De Maesschalck, D. Colle, I. Lievens, M. Pickavet, P. Demeester, C. Mauz, J. Derkacz, Pan-european optical transport networks: an availability-based comparison, Photonic Netw. Commun. 5 (3) (2003) 203–225.

[51] A.J. Gonzalez, B.E. Helvik, Characterisation of router and link failure processes in UNINETTs IP backbone network, Int. J. Space Based Situated Comput. 7 (1) (2012) 3–11. 2

[52] S. Kaplan, B.J. Garrick, On the quantitative definition of risk, Risk Anal. 1 (1) (1981) 11–27.

[53] B. Chandrasekaran, T. Benson, Tolerating SDN application failures with legoSDN, in: Proceedings of the 13th ACM Workshop on Hot Topics in Networks, ACM, 2014, p. 22.

[54] POX Wiki, [Online]. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki.

[55] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, Advanced study of SDN/OpenFlow controllers, in: Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, ACM, 2013, p. 1.

[56] W.Y. Huang, J.W. Hu, S.C. Lin, T.L. Liu, P.W. Tsai, C.S. Yang, J.J. Mambretti, Design and implementation of an automatic network topology discovery system for the future internet across different domains, in: Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on, IEEE, 2012, pp. 903–908.

[57] Att/pox, Accessed on July. 15, 2019. [Online]. Available: https://github.com/att/pox/blob/master/pox/openflow/discovery.py.

[58] D.A. Schult, P. Swart, Exploring network structure, dynamics, and function using networkx, in: Proceedings of the 7th Python in Science Conferences (SciPy 2008) (Vol. 2008, 2008, pp. 11–16.

[59] A. Malik, B. Aziz, M. Adda, Towards filling the gap of routing changes in software-defined networks, in: Proceedings of the Future Technologies Conference, Springer, Cham., 2018, pp. 682–693.

[60] R. Bhandari, Survivable Networks: Algorithms for Diverse Routing, Springer Science & Business Media, 1999.

[61] S. Poretsky, B. Imhoff, K. Michielsen, Terminology for benchmarking link-state IGP data-plane route convergence (no. RFC 6412), 2011.

[62] SNDlib library, [Online]. Available: http://sndlib.zib.de.

[63] A. Medina, A. Lakhina, I. Matta, J. Byers, 2001, BRITE: an approach to universal topology generation. In: Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on (pp. 346–353). IEEE.

[64] B.M. Waxman, Routing of multipoint connections, IEEE J. Sel. Areas Commun. 6 (9) (1988) 1617–1622.

[65] SDN proactive fault handling, Accessed on July. 15, 2019. [Online]. Available: https://github.com/Ali00/SDN-Smart-Routing.

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

[67] L. Saino, C. Cocora, G. Pavlou, A toolchain for simplifying network simulation setup, in: Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2013, pp. 82–91.

[68] M. Ohring, J.R. Lloyd, Reliability and failure of electronic materials and devices, Academic Press, 2009.

[69] ZeroMQ, [Online]. Available: http://zeromq.org/.

Ali Malik is a Postdoctoral Researcher at the School of Electrical and Electronic Engineering, Technological University Dublin, Ireland. He received his B.Sc. degree in computer science from Al-Qadisiyah University, Iraq, in 2009. He also holds an M.Sc. degree in information technology from BAMU University, India, in 2012. He obtained his Ph.D. degree in computer science from the University of Portsmouth, United Kingdom, in 2019. His current research interests include software-defined networks, routing, fault management, risk and cybersecurity.

Benjamin Aziz is a Senior Lecturer in Computer Security at the School of Computing, University of Portsmouth, United Kingdom. He holds Ph.D. degree in formal verification of computer security from Dublin City University (DCU), Ireland, in 2003. He has worked in the past as a postdoctoral researcher at Imperial College London and Rutherford Appleton Laboratory, in areas related to security engineering of large-scale systems, formal design and analysis, requirements engineering and digital forensics. He is on board program committees for several conferences and working groups, such as ERCIM's FMICS, STM, Cloud Security Alliance and IFIP WG11.3. His research interests include formal modelling, security, computer forensics, risk management, software engineering, IoT and software-defined networks.

Mo Adda is a Principal Lecturer in computer networks at the School of Computing, University of Portsmouth. He received the Ph.D. degree in distributed systems and parallel processing from the University of Surrey. He was a Senior Lecturer with the University of Richmond, where he taught programming, computer architecture, and networking for ten years. From 1999 to 2002, he was a Senior Software Engineer developing software and managing projects on simulation and modelling. He has been researching parallel and distributed systems since 1987. His research interests include software-defined networks, wireless sensor networks, mobile networks and business process modelling, simulation of mobile agent technology and security.

Chih-Heng Ke is an Associate Professor with the Department of Computer Science and Infor- mation Engineering, National Quemoy University, Kinmen, Taiwan. He received the B.S. and Ph.D. degrees in electrical engineering from National Cheng-Kung University in 1999 and 2007, respectively. His current research interests include multimedia communications, wireless networks, and software-defined networks.