

Software-Defined Networks: A Walkthrough Guide From Occurrence To Data Plane Fault Tolerance

Ali Malik¹, Benjamin Aziz¹, Ali Al-Hajj¹, and Mo Adda¹

¹University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, Portsmouth PO1 3HE, United Kingdom

Corresponding author:

Ali Malik¹

Email address: ali.al-bdairi@port.ac.uk

ABSTRACT

In recent years, the emerging paradigm of software-defined networking has become a hot and thriving topic that grabbed the attention of industry sector as well as the academic research community. The decoupling between the network control and data planes means that software-defined networking architecture is programmable, adjustable and dynamically re-configurable. As a result, a large number of leading companies across the world have latterly launched software-defined solutions in their data centers and it is expected that most of the service providers will do so in the near future due to the new opportunities enabled by software-defined architectures. Nonetheless, each emerging technology is accompanied by new issues and concerns, and fault tolerance and recovery is one such issue that faces software-defined networking. Although there have been numerous studies that have discussed this issue, gaps still exist and need to be highlighted. In this paper, we start by tracing the evolution of networking systems from the mid 1990's until the emergence of programmable networks and software-defined networking, and then define a taxonomy for software-defined networking dependability by means of fault tolerance of data plane to cover all aspects, challenges and factors that need to be considered in future solutions. We discuss in a detailed manner current state-of-the-art literature in this area. Finally, we analyse the current gaps in current research and propose possible directions for future work.

1 INTRODUCTION

COMPUTER networks play an essential role in changing the lifestyle of modern society. Nowadays, most of the Internet services are located in data centers, which consist of thousands of computers that are connected via large-scale data center networks. Typically, wide-area networks interconnect data centers that are distributed across the globe. Internet users are usually using their devices (i.e. computer, mobile, tablet, smart watch etc.) to access the various services available on the Internet through different means; such as WiFi, Ethernet and cellular networks. Managing the networks efficiently to meet the requirements of the Quality of Service (QoS) and the Service Level Agreements (SLA) are the core challenging points of computer networks, which need to be improved continuously due to the increasing number of devices that are connected to the Internet, which is currently estimated to be over 7 billion devices and is expected to be more than double that before 2020 Melcherts (2017); Mourtzis et al. (2016).

With its billions of users and diverse networks, many pundits are now considering the Internet as the future infrastructure of global information. Beside this view, it has been widely debated about the pricing in computer networks whether it's on the basis of the performance and the quality of provided services, which is also called "flat pricing", or to basing charges on actual usage, which is also called "usage-based pricing", more details can be found in (Shenker et al., 1996).

Since most computer networks are commercial enterprises, operators are also willing to maximise revenue and profits and therefore, their networks confined to meet some business requirements (i.e. QoS) and benefits. Network failures (e.g. link failure) usually result in service interruption, which will lead to SLA violation as a consequence and hence revenue loss. Therefore, resilience methods are very important

to guarantee the high network availability and consequently avoid revenue losses.

Fault tolerance is a crucial property for computer networks availability, which has been widely explored since the 1950s (Peter and Thomas, 1981). However, this concept needs to be revisited from time to time in order to explore and revise the new networking solutions and technologies. In this context, the recently emerged paradigm of software-defined networks (SDNs) gives a glimpse of hope for a new networking architecture that provides more flexibility and adaptability. While, fault tolerance is considered one of the key concerns with respect to SDNs dependability. Although some studies have been undertaken to identify the limitations of SDNs fault tolerance, in this paper we mainly focus on the data plane fault tolerance issues of SDNs by providing a comprehensive literature survey on SDN data plane recovery strategies to date in addition to briefly summarising the historical development of programmable networks.

The reminder of the paper is organised as follows. In Section 2.1, an overview on traditional network systems and contemporary strategies of data transmission in such systems is given. Section 2.2 presents an intellectual history of programmable networks and the strenuous efforts made by researchers since the mid of 90's. Section 2.2.1 provides the main concepts of SDNs and how it differs from traditional networking systems, followed by an approximated number of SDN articles that have been published so far (Section 2.2.2). Section 3 outlines the dependability indices and then, a comprehensive survey has been undertaken to find achievements and challenges in SDN fault tolerance. A summary of main issues are categorised and presented in Section 4. Finally, we have listed some open research challenges that need to be addressed in Section 5, which somewhat gives future directions. The paper is concluded in Section 6.

2 EVOLUTION OF COMPUTER NETWORKS

2.1 Traditional Networks

Traditionally, the distributed control systems in networking devices along with a set of defined protocols (e.g. OSPF (Moy, 1998) and RIP (Hedrick, 1988)) constitute a fundamental technology that has been adopted to send and receive data via networks around the world in recent years. The OSPF protocol, which is usually run periodically by every single switch, is responsible for collecting the information about the state of the local links in the network, so that the collected information will be processed by every switch in the network in order to calculate the routing table. In case of failure events (e.g. link failure), the protocol will propagate an update regarding the failed link(s), so as to amend the set of routes within seconds. While, over the past few decades, these protocols have increased the inflexibility of network management by making the network operators lose their visibility over their networks (Benson et al., 2009). The reason for this is that the switches were like a black box and the opaqueness of protocols, which are written by multiple vendors, prevented the network operators from modifying these proprietary implementations in order to satisfy their routing requirements and goals.

The rigidity of the traditional network architecture increases the difficulty of handling the network transferred data. The reason is that the conventional network switches are vertically integrated by means of the two functional components, namely *control plane* and *data plane*, which are bundled together inside the networking forwarding elements. Switches discover the network topology by relying on the distributed protocols that are running in the control plane, while the data plane is responsible for forwarding the incoming packets (i.e. traffic) based on the decision made by the control plane. The network operator had to set up each switch manually through the configuration CLI, in which hundreds of parameters need to be adjusted in order to orchestrate the work of all the control plane's running protocols. Therefore, the network operators had neither a high level of abstract view on their networks, nor a high level of interface abstraction to configure the huge number of parameters. Hence, the attitude of networks was very challenging and opaque from the perspective of the network operators and the transition from IPv4 to IPv6 is a perfect illustration of that, since it was started more than 10 years ago and still the protocol update is incomplete for a large number of networks. All these inconvenient obstacles made the network operators keen to get rid of the rigidity of the proprietary implementations by replacing them with a new flexible architecture. A solution with more flexibility is not only of benefit to the network operators, but also for the permanency of the Internet services, since it was stated in (Lin et al., 2011) that ossification of the Internet widely expected due to the massive number of connections and low degree of manageability.

2.2 Programmable Networks

The management of the legacy networks is complex and difficult due to the nature of the networking equipment, from switches to middleboxes such as firewalls, intrusion detection and load balancers. That is work based on distributed protocols and requires a long process of configuration across every single device. This mode of operation has increased the level of complexity of networking system management and, therefore, the network administrators were looking for more programmable and open solutions to enable them to build agile networks instead of the traditional rigid system. According to (Feamster et al., 2014), the history of programmable networks can be divided into three stages:

1. *Active Networks (1994-2000)*: In the beginning, the researchers were designing protocols and testing them in their labs which simulate a large scale network. Then, the new solutions were submitted to the Internet Engineering Task Force (IETF) for standardisation of the new protocols. Usually, the protocols standardisation process was very slow which discouraged the enthusiasm of researchers. Therefore, this situation pushed the researchers to think about opening up network control with a view by adding a programmable interface (or network API). The initial idea was described in (Tennenhouse and Wetherall, 2002) and the main aim was to split the services from the underlying structure and hence accelerate the innovation. This was a radical approach at that time because it changed the strategy of the traditional networking systems. Additionally, many of the Internet pioneers were against simplifying the core of the network, as they thought this affect the success of the Internet itself. The ideas behind active networks were firmed up by the program of the U.S. Defense Advanced Research Projects Agency (DARPA), which discussed the future of networking systems and identified the barrier to innovations, distributed protocols and performance issues (Tennenhouse et al., 1997). Two main programming models were suggested during the era of the active network:
 - a) *The Capsules model*, which is a mobile code that carried in-band in data packets and should be executed in the active nodes (Wetherall et al., 1998).
 - b) *The programmable switch/router model*, where the software-programmable elements of every switch/router were able to interpret messages as programs and hence codes were executing at nodes in an out-of-band manner (Smith et al., 1996; Bhattacharjee et al., 1997).
2. *Control-Data separation Networks (2001-2007)*: In the early 2000s, the Internet traffic grew rapidly, hence there was a requirement for more flexibility over some networking services, such as network reliability and predictability. For this reason, the network operators were looking for a new solution that was in line with the new challenges. This solution must consider the tightly coupled architecture between the data and control planes of the conventional routers and switches as it accelerates the task of network management. In consequence, that period experienced numerous efforts that addressed methods of separating the data and control planes. ForCES (Forwarding and Control Element Separation) (Dantu et al., 2004) is one of the most significant solutions that addressed the challenge of the open interface between the control and data planes, which is also standardised by the IETF. The Routing Control Platform (RCP) (Feamster et al., 2004; Caesar et al., 2005) is another radical solution that focuses on facilitating the logically centralised control of the network through separate routing from routers. Furthermore, the SoftRouter architecture (Lakshman et al., 2004) has separated the functions of the control plane from the data packet forwarding through utilisation of the ForCES API. However, unfortunately, most of the manufacturing companies did not support the new approaches (such as the ForCES) and that hampered wide deployment. Afterwards, researchers started exploring new clean-slate architectures, since the dominant vendors had little incentive to adopt the previous solutions. The 4D project (Greenberg et al., 2005) reshaped the network architecture into four planes, namely: *decision*, *dissemination*, *discovery* and *data* planes, with the purpose of centralising the network functions of control and management. Ethan (Casado et al., 2007) (an extension of SANE (Casado et al., 2006)) has addressed some of the 4D considerations by using two components: (i) a centralised controller that is supposed to manage the global network security policies and (ii) Ethan switches that receive the forwarding rules from the controller. There were two main limitations that prevented Ethan from being adopted: firstly, knowledge about on network nodes and users is missed out; secondly, it requires control over routing at the flow level. However, these two limitations were addressed by the NOX network operating system (Gude et al., 2008).

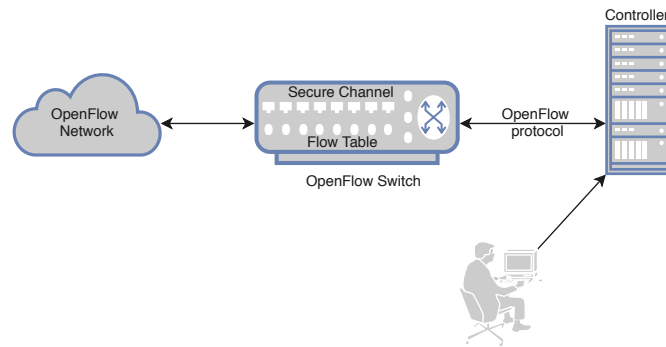


Figure 1. Basic Architecture of OpenFlow

3. *OpenFlow networks (2007-2010)*: The SANE and its successor Ethan (the simple switch design of Ethan in particular) set the stage for the inception of the OpenFlow era. A research group from Stanford University has developed the OpenFlow (OF) protocol (McKeown et al., 2008) as a clean slate technology that deployed in the university campus network. OpenFlow acts as an API that remotely control the forwarding tables of the network elements (i.e. switches and routers). Figure 1 illustrates the OpenFlow architecture in which the network elements are managed by one or multiple controllers. The OpenFlow enabled switch consists of two main components: (i) a *secure channel* as an interface to establish the connection over an encrypted channel (usually via SSL and/or TLS) between the controller and the switch, and (ii) a *flow table* (could be one or multiple) that host a number of flow entries (i.e. forwarding rules) which are necessary to process the incoming packets. Each OF switch contains at least one flow table and each flow table contains one or multiple flow entries, sorted by priority order, as illustrated in Figure 2. Each flow entry maintains of the following three main components:

- *Matching Fields*: To match against the incoming packets, this field includes information such as source and destination ports, IP/MAC address and VLAN ID.
- *Actions/Instructions*: To specify how the matched packet(s) will be handled, which is typically to forward the packet (either to one of the switch's ports or to the controller) or to drop.
- *Stats*: To include statistics for each flow such as the number of packets/bytes that matched a particular entry and time since last packet matched the rule.

Currently, OpenFlow protocol supports a wide range of various types of OpenFlow messages that can be used to coordinate the forwarding devices and some of the essential ones are listed as follows:

- **HELLO**: This message is launched by either switch or controller at the point when the connection is established. This message carries the information about the highest version

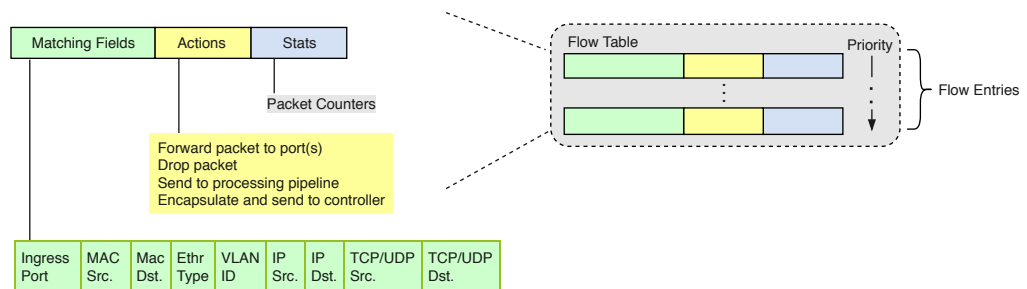


Figure 2. Flow table and flow entries

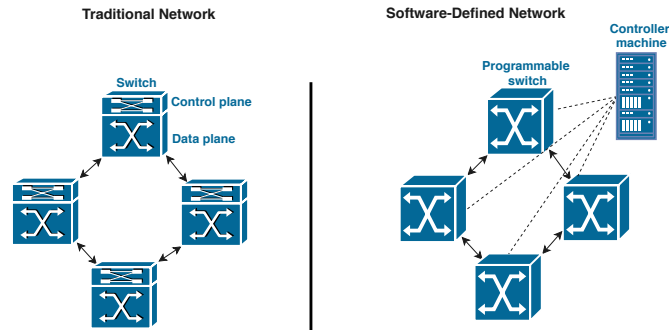


Figure 3. Traditional network versus SDN architecture

that can be supported by the sender as stepping stone with the receiver towards building a channel between them.

- **ECHO:** This message is used to check operational status and liveness between the controller and switches or routers. For instance, the network controller can send an **ECHO-REQUEST** to a switch and the particular switch in turn will respond to the controller request by sending an **ECHO-RESPONSE**. When **ECHO-REQUEST** fails to get a reply, controller will report this as disconnection status (e.g. failure).
- **PACKET-IN:** This message enables OpenFlow switches to send data packets to the controller. There are two occasions that might lead a switch to generate this message; it could be either an explicit action as a result of a match asking for this behaviour, or when the received data packet fails to match any rules of the flow table.
- **PACKE-OUT:** As a response upon receiving a **PACKET-IN** message, the controller initiates a **PACKE-OUT** in which the instruction on how to deal with the received packet is encapsulated and send to the switch.
- **FLOW-MOD:** This message enables the controller to update the flow table states of any particular switch in the network. This message can be used either reactively or proactively to modify the rules (e.g. add, delete, modify).
- **PORT-STATUS:** This message is sent by switches directly to the controller as an announcement that there will be a port status change (i.e. port up, port down), which is necessary for the controller to deal with failure events.

The OpenFlow is considered to be the father of SDN due to the wide adoption and smooth deployment that crystallised the principle of SDN as a pragmatic and bold view.

2.2.1 Software-Defined Networks

The lessons learned from the past were mainly related to easing the network management and control and to make the networking systems more elastic to open the door for new innovations. In this regard, Software-Defined Networking (SDN), which is often linked to OpenFlow, has emerged as a promising solution to tackle the inflexibility of the previous networking systems. The success of OpenFlow, which has gained huge attention from both academia and commercial industry, has driven the development of SDN as a next generation of networking system architecture. Figure 3 demonstrates the difference between the architecture of SDN and the traditional networks. Such a new networking paradigm of SDN with much more flexibility compared to the traditional networks meant that SDNs are nowadays adopted by many of the well known pioneering companies such as Deutsche Telekom, Google, Microsoft, Verizon and CISCO. These have recently combined in 2011 to launch the Open Network Foundation (ONF) (ONF, 2018), which is a nonprofit consortium that aims to accelerate the adoption of SDN technologies. In SDN the data and control planes have been physically detached resulting into two isolated layers as follows:

Table 1. Some of SDN controllers that supports OpenFlow

Controller	Developer	Implementation	Open-Source
NOX (Gude et al., 2008)	Nicira	C++	Yes
Maestro (Cai et al., 2010)	Rice Uni.	Java	Yes
POX (McCauley, 2012)	Nicira	Python	Yes
Floodlight (Project Floodlight, 2012)	Big switch	Java	Yes
Trema (Trema, 2011)	NEC	Ruby/C	Yes
Beacon (Erickson, 2013)	Stanford Uni.	Java	Yes
OpenDaylight (OpenDaylight, 2013)	Linux Foundation	Java	Yes

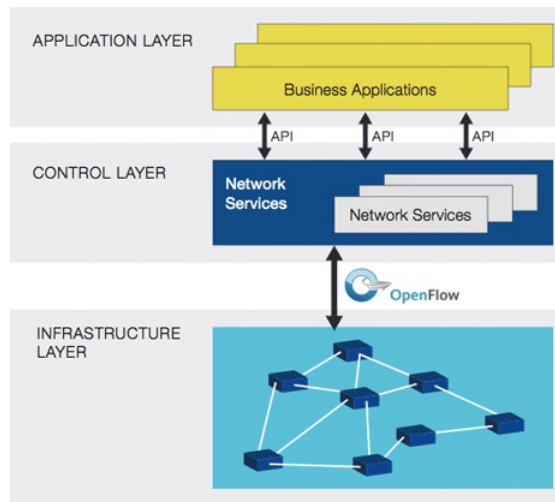


Figure 4. Overview of SDN layers (ONF Market Education Committee, 2012)

Control Layer

The creation of the OpenFlow protocol is accompanied with development of some controller platforms such as those illustrated in Table 1: The control layer/plane, which is usually called the *controller*, represents the network brain as it provides a global view over the whole network and enables the network operators to configure or reconfigure their infrastructure dynamically by customizing policies/protocols across the network forwarding elements. Control plane acts as an intermediary between the network applications and the infrastructure layers as depicted in Figure 4. Typically, the network controller communicates with the application layer via the northbound interface and governs the network switches via the southbound interface. Controllers usually offer APIs, which represent the northbound interface, for the programmers to develop various kinds of applications and to abstract the low-level instruction sets of the southbound interface. The southbound interface defines the communication protocol between the controller and the switches, furthermore, it provides an API that is essential to define the instruction set of the infrastructure layer elements. Most of the current SDN controllers support and utilise the OpenFlow protocol as a southbound API, however, a few other controllers (such as the OpenDaylight) support a wide arrange of southbound APIs in addition to the OpenFlow.

A single SDN controller is able to process a surprising number of flow requests that could reach and even exceed a million per second. Although, one controller is sufficient to handle and manage a large-scale network, multiple controllers may be required to increase the network availability in case single controller fails. For a survey on SDNs with multiple controllers, we refer the interested readers to (Zhang et al., 2017b).

Infrastructure Layer

The infrastructure layer, which is also known as the *Data plane*, consists of a set of forwarding devices (i.e. switches and routers) that are interconnected via either wireless channels or wired cables and constitute the network topology. The decoupling between the *control* and *data* planes

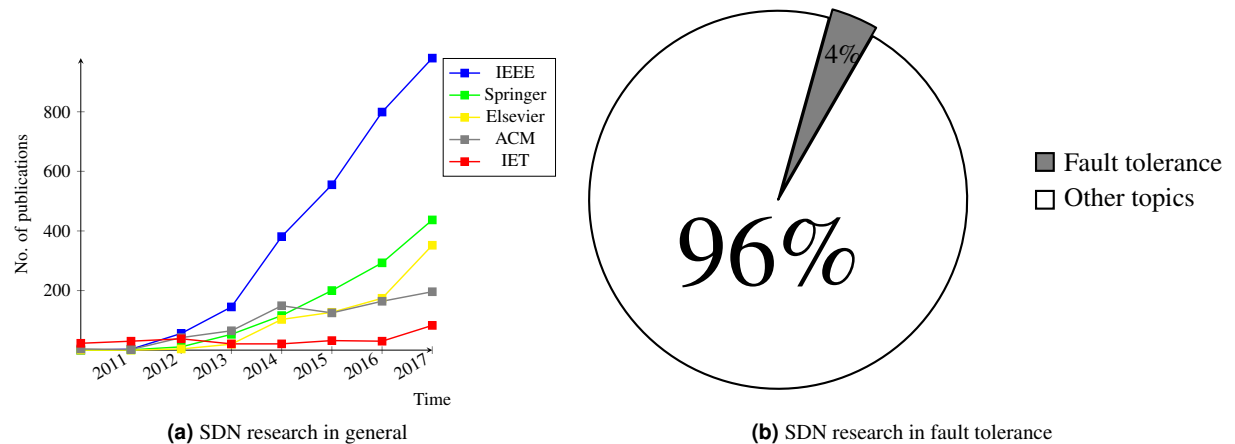


Figure 5. Amount of SDN published research between 2010-2017.

results in dumb forwarding elements whose packet forwarding behaviour is dictated by the controller. Hence, the newly arriving packets, that have no matching rules yet, will be forwarded to the controller (i.e. PACKET-IN) that, in turn, sends the pertinent information (i.e. PACKET-OUT) as a forwarding rule to the relevant switch(es) over the southbound interface, which eventually will be installed as a flow entry. Currently, several OpenFlow-enabled devices are available on the market in both hardware (i.e. native) and software (e.g. run in testbed and virtualisation) based implementations (Nunes et al., 2014). It is also worth mentioning that most of the current switches have a relatively limited space on the ternary content-addressable memory (TCAM), which can be reached up to 8000 entries, however, some of the recent OpenFlow appliances such as Gigabit Ethernet and EZchip NP-4 support up to 32000 and 125000 entries, respectively (Kreutz et al., 2015). It is obvious that the relatively small TCAM might cause a scalability issue for the future deployment of SDN.

• Application Layer

The network application layer comprises a set of network applications that are essential to implement the control logic for the whole network domain. SDN applications run on top of the controller and utilise the northbound interface to request the network state in order to manipulate the provided services. The applications outcome will be translated into instructions and then sent to the forwarding elements in the infrastructure layer via the southbound interface. This simplified the deployment of new network protocols and services and despite the various aims of SDN applications, according to (Kreutz et al., 2015) it can be classified into five categories; traffic engineering, mobility and wireless, measurement and monitoring, security and dependability and data center networking.

2.2.2 Research Trend of Software-Defined Network

After the development of the OpenFlow protocol during the period 2007-2010 (as mentioned in Section 2.2), exploring the pros and cons of the different aspects of the new network architecture; such as the interoperability, applicability and dependability, was the focus of attention from the second quarter of the last decade. This section shows the amount of SDN conducted research from several academic publishers in the field of computer science and engineering like IEEE, ACM, Elsevier, Springer and IET. We have collected an approximate number of research publications (i.e. journals, conference proceedings and corporate technical papers) that were published in the period 2010-2018, since works that predate the usage of the term "SDN" are not reported. For example, some synonyms such as programmable and OpenFlow networks were used instead of software-defined networking. On one hand, Figure 5a illustrates the increasing number of research efforts focusing on SDN over the last 10 years. According to the figure, more than 7200 research publications over the last 10 years were published just from the aforementioned publishers. Table 2 shows the total number of published items. On the other hand, Figure 5b shows the

Table 2. Publisher-based classification

Publisher	No. of SDN research
IEEE	3386
Springer	1559
Elsevier	1109
ACM	854
IET	315

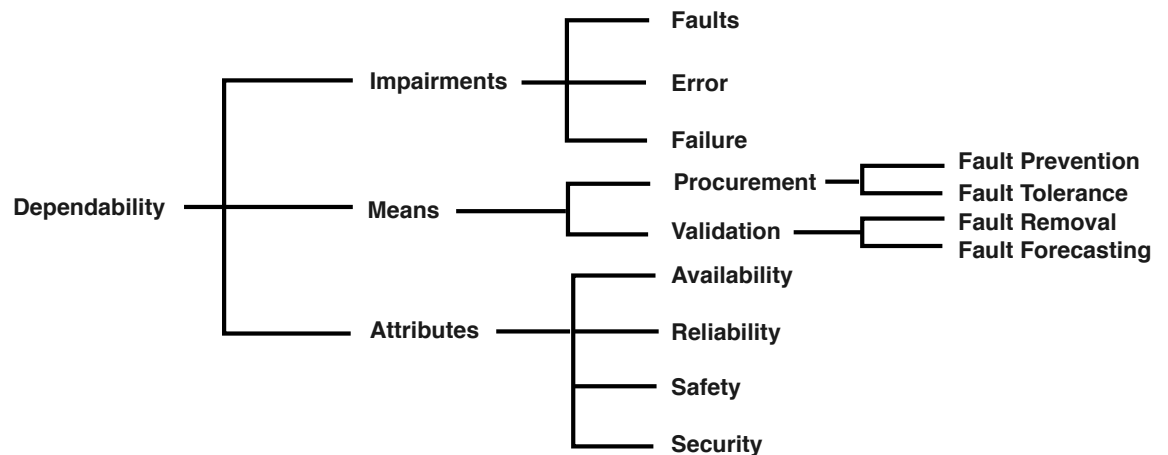


Figure 6. Dependability tree (Laprie et al., 1992)

relative size of SDN fault tolerance research, which has been filtered on the basis of two keywords; namely *fault tolerance* and *software-defined networking*, as a percentage of the entire scope of SDN research.

3 DEPENDABILITY AND FAULT MANAGEMENT

The rapidly increasing number of critical business applications and secured data transmission over modern day communication networks brings up the demand for a high degree of robustness and reliability. Apparently, communication networks are prone to either unintentional (unplanned) failures due to various causes such as human errors, natural disasters (e.g. earthquakes), overload, software bugs and so on, or to intentional (planned) failures that are caused by the process of maintenance (Markopoulou et al., 2004, 2008). All network elements (e.g. forwarding devices and links) are susceptible to failure incidents and in such cases some network facilities (like routing) will be harmed. In addition, failures could cause a financial loss to the service providers (e.g. cloud service providers), for instance, according to the statistics of 28 cloud providers (from 2007 to 2013), financial losses were estimated at approximately \$285 million (USD) as a consequence of infrastructure and application failures (Cérin et al., 2013).

Therefore, a failure recovery scheme is a necessary requirement for networking systems to ensure the reliability and service availability. In computing systems, according to (Laprie et al., 1992), *dependability* is the umbrella concept that subsumes a set of attributes such as reliability, availability and fault-*{tolerance, prevention, prediction and removal}* as illustrated in Figure 6. In other words, dependability includes a range of properties that aim to mitigate threats through different means. Fault management is the expedient to which the concept of dependability is attainable. Hence, the efficiency and efficacy of the fault management functions impact upon dependability. This thesis is mainly focussed on *fault tolerance*, which is the property that enables the system to continue functioning properly even in the presence of failure in some of its components. Fault tolerance has been a widely used approach in communication networks to recover from failure when it occurs. The main concept is to mask the failed (unhealthy) components by relying on and utilising some ready/available (healthy) ones. In what follows, the essential

failure recovery techniques of paths in both conventional networks (in brief) and SDNs (in detail) are discussed and the pros and cons of each technique are elucidated.

3.1 Recovery in Legacy Networks

In general, failure recovery mechanisms of carrier-grade networks are categorized into two types: *protection* and *restoration*. In protection, which is also known as *proactive*, the alternative solution is preplanned and reserved in advance (i.e. before a failure occurs). According to (Vasseur et al., 2004), there are a couple of protection schemes that can be applied to recovery from network failure, these are:

- One to One (1:1): in which, one protection path is dedicated to protect exactly one path.
- One to Many (1:N): in which, one protection path is dedicated to protect up to N paths.
- Many to many (M:N): in which, M specified protection paths are dedicated to protect up to N working paths such that $M \leq N$.

In contrast, in restoration, which is also called *reactive*, the possible solution is not preplanned and will be calculated dynamically when failure occurs. Despite the different extant schemes of failure recovery, there are some shared succession phases in between, these are, the *recovery* and *reversion cycles* (Vasseur et al., 2004).

On the one hand, at the moment of failure (i.e. when failure occurs), a *fault detection* process will take some time to detect the failure and it typically depends on the speed of the fault detection mechanism. For example, there are different mechanisms that can be used to detect the link failure such as Bidirectional Forwarding Detection (BFD) (Katz and Ward, 2010), which detects the failure through the periodic exchange of monitoring messages via an established session between source-destination nodes. Also, the Loss of Signal method (LoS) (Vasseur et al., 2004), which detects the failure by depending on the port state of the device (i.e. down or up), however, the frequency of signals sent varies. Once the fault detection process is finished, the node that detected the failure will wait some time, that is the *Hold-off time*, before propagating the announcement about the detected failure. This is because some kind of failure such as a cable cut might be rapidly repaired in some IP networks that are supported by an optical transport layer. When the fault still persists after the period of the *Hold-off*, a failure message announcement (i.e. *Fault notification*) will be sent throughout the network to notify the relevant nodes that they should perform a recovery action. The later step, which is the *Recovery operation*, depends on the recovery scheme used (i.e. protection or restoration). Finally, the traffic will start flowing on the new path after the last recovery action and it is usually affected by some factors such as the: location of failure, delay of recovery process, scheme of failure recovery. Figure 7 (a) illustrates the cycle of recovery. On the other hand; and after the termination of the recovery cycle, the network will pass into a fully operational state. However, the resulting paths from the recovery cycle may be less ideal than before the failure (e.g. the alternative path is longer than the original). Therefore, when the fault is fully repaired, a switch back to the original path is important in order to keep the network in an optimal state. The operation of switching back to the former path is called *revertive* and Figure 7 (b) shows the phases of the reversion cycle, which bears a strong similarity to the cycle of recovery. When the fault is repaired, it could take some time (i.e. *fault clearing time*) to be detected. Afterwards, the repaired notification might be delayed for a while (i.e. *Hold-off time*) and this is necessary for the network stability, especially when the fault is intermittent. After that, the notification about the repaired fault will be propagated throughout the network so that the relevant nodes will be notified to deploy the *reversion operation*. Finally, the traffic will start flowing through the original working path after some time (i.e. *traffic reversion time*), which is the time between the last reversion action (i.e. last time of using the alternative path) and the restoration of the ideal/original path. In fact, the traffic reversion time is small because both paths are working and ready to be utilised.

In contrast to the recovery cycle—which is a mandatory operation that responds to unforeseen failures as soon as they are detected (the quicker the better), the reversion cycle is another essential operation but it can be planned in advance to avoid the service disruption (quick but not hasty).

3.2 Recovery in Software-Defined Networks

The new features of SDNs such as the global view of the central controller and the programmable interfaces have changed the traditional methods of fault management from decentralise, which mainly relies on the distributed protocols, to centralise, with the view of fixing today's fault management issues

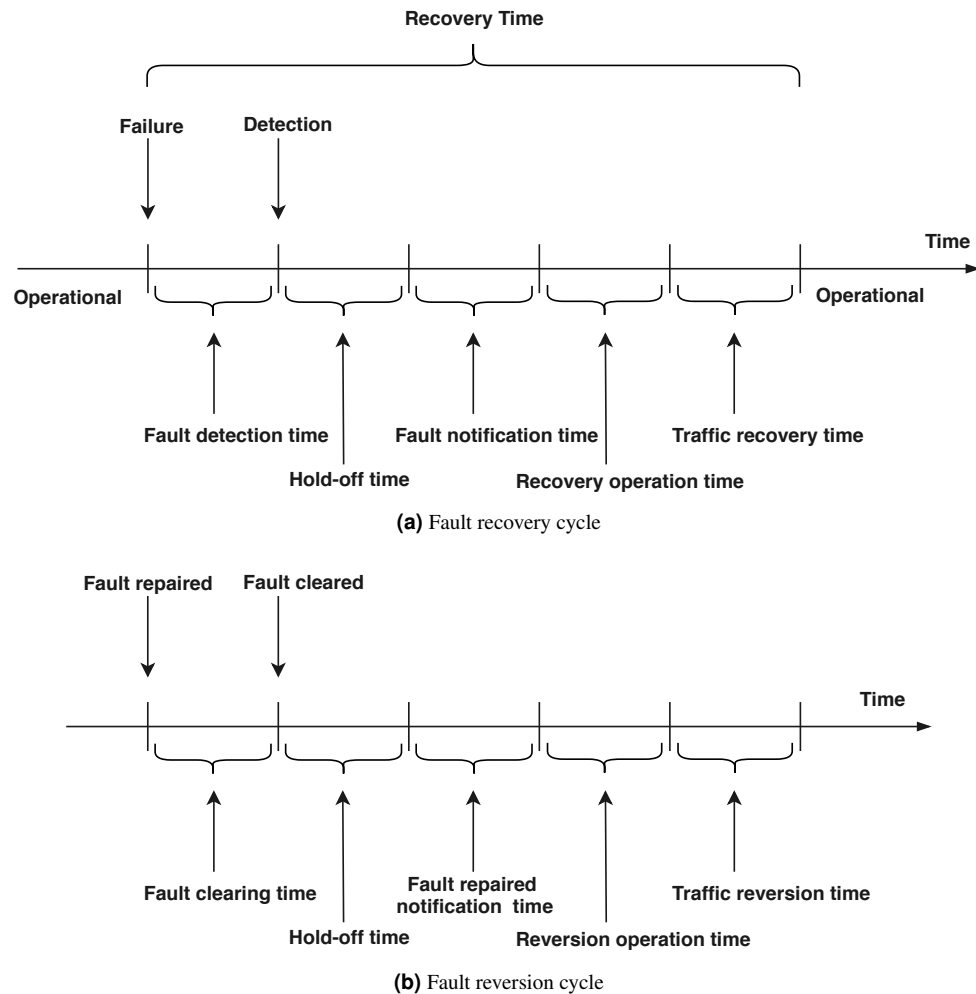


Figure 7. Recovery process in legacy networking systems, (Vasseur et al., 2004).

(e.g. fault localization). However, the new architecture of SDN imposes some constraints and concerns on the implementation of the failure recovery schemes. Since the SDN consists of three layers, then each layer is considered as a point of failure, as follows:

- *Control layer*: The controller dependability is of critical importance due to the fact that SDN is a centralised networking system and the controller is responsible for managing the network events and activities. Therefore, when failure occurs in the controller, then all network services will be halted.
- *Application layer*: Failure can be also generated due to an application bug, which will lead to a network misconfiguration and error. However, sometimes the application layer and control layer are considered as one layer (i.e. control plane) and hence it can be considered as a control plane failure again.
- *Data layer*: A failure in the network infrastructure elements (i.e. switches and links) will lead to service degradation where this will directly affect the QoS and significantly impact the network users.

Therefore, the different layers of SDNs meant that the fault tolerance of each layer has to be built separately as the failure that could occur in a particular layer may or may not affect the other layers. For example, link failure may not affect the control layer, while, the controller failure may affect the entire infrastructure layer. Thus, it is possible to categorise the SDN fault tolerance into the following tiers:

3.2.1 Fault Tolerance for Control Plane

Basically, a single centralised control is sufficient to manage all the switches of the data plane. However, reliance on a single controller may not be wise for two reasons: *first*, the controller is a single point of failure and *second*: for large-scale networks, a single controller is a potential performance bottleneck. Thus, a number of studies have focused on this issue with a wide range of solutions such as: a backup controller (Sidki et al., 2016), a broker-based approach for multiple controllers (Kurtz and Wietfeld, 2017) and distributed controller clusters (Abdelaziz et al., 2017). Given that this paper is mainly focussed on the fault tolerance of the data plane, then the aforementioned short overview regarding the issues of fault tolerance in the control plane will be sufficient.

3.2.2 Fault Tolerance for Data Plane

Since link and node failure is an issue almost as old as computer networks, so far, SDN follows the traditional fundamental strategies of failure recovery (i.e. protection and restoration) to recover from the data plane failures. However, the fault management in SDNs differs from the legacy networks in its mean of computing and updating the routing tables which react to failure incidents. Traditionally, the distributed networking appliances (e.g. switches) are responsible for building the routing table and reconstructing when link failure is detected. In contrast, in OpenFlow networks the routing decisions for each forwarding element (e.g. switch) are dictated by the central controller. Based on the global view that the controller possesses, any change in the network topology (e.g. when failure occurs) will lead the controller to immediately reconfigure the flow tables of the switches that are involved in the affected routes either in a direct (controller-needed) or indirect (controller-needless) way. This section is subdivided into two subsections in order to introduce some existing methods in two different aspects, namely protection and restoration. We categorised the proposals into two categories; namely, *native* and *hybrid*. We call the newly designed solutions of the literature with *native solutions*, however, the tailored solutions that derived from, or combined with, the traditional networking systems as *hybrid solutions*. Figure 8 presents the four possible cases of fault tolerance schemes against the relationship with the SDN controller.

		Fault Tolerance	
		Proactive	Reactive
SDN Communication Mode	Controller-needless		
	Controller-needed		

Figure 8. Classification of SDN recovery schemes

Protection

The method of protection will be employed after the failure is detected. As mentioned earlier the protection mechanism is capable of masking a failure proactively by calculating the recovery settings (i.e. routing tables) at an early stage, in other words, the backups are pre-established. This means that each switch will reserve the forwarding information for both primary and backup (i.e. protection) paths.

A) Native solutions

In (Sgambelluri et al., 2013b) and (Sgambelluri et al., 2013a), the authors produced a fast recovery mechanism in case of single link failure, in which the controller installs low priority flow rules for the backup paths along with the primary ones. When link failure occurs, the two switches connected to the failed link will delete the primary path information and redirect the traffic to the backup one. In contrast, when the failed link is physically repaired, the switches will send a notification message, OFPT_FLOW_RESTORE, as an extended feature to the OpenFlow protocol, to the network controller in order to re-install the primary path. The experiments, with small scale topologies, show that the recovery can be achieved within less than 64 μ s.

Kitsuwan et al. (2015) produced the Independent Transient Plane (ITP) as a new protection scheme, which is originally proposed in (Kitsuwan et al., 2014), for fast rerouting with a view of reduce the number of flow entries that are required in the rerouting operation. ITP consists of two completely independent planes, namely *working* and *transient* planes. The working plane is used for routing under the normal operation of the network (i.e. no failures), which also contains the information on the backup disjoint path for each of the source-destination nodes in the network. In contrast, when a link failure occurs, the packets will be routed over the transient plane until the failure is repaired. Controller intervention is required for switching from the working to the transient after detecting a link failure. Compared with the work of (Sgambelluri et al., 2013b), this study demonstrates how the number of flow entries can be reduced by 50% in topologies with up to 30 nodes.

Some studies have utilised the *fast failover* feature of OpenFlow v1.1 (ONF, 2011), in which the OpenFlow switch has been extended to include a new table (i.e. Group table) in addition to the Flow table. Again, the Group table encompasses a set of group entries, these are *ID*, *type*, *counter* and *action buckets*, where the last are ordered actions that are associated with specific ports and are to be executed according to the ports alive stats. Sharma et al. (2013) have produced a 1:1 protection mechanism using the *fast failover* feature to handle the link failures. According to their study, the controller computes the disjoint paths for the network traffic and then installs them as protection paths in the switches. When link failure occurs, the switch will activate the failover rules and redirect the affected traffic to the backup protection paths without requiring the controller's intervention. The study showed that the carrier-grade requirement to recover from failure within 50 μ s can be achieved with the proposed protection method.

Van Adrichem et al. (2014) argued about the time required to detect the link failure and the time required to compute the alternative path. The authors introduced (i) a per-link BFD session (instead of per-path) as a fast failure detection method and (ii) a fast failover mechanism as a protection method against failures. The experiments showed that the recovery time can be achieved within sub 50 μ s, which meets the carrier-grade requirements.

Ramos et al. (2013) proposed SlickFlow, a source-based routing method to enhance the scalability and fault tolerance in OpenFlow networks. Since the packet header provides an additional limited segment of information that can be used for the purpose of adding some details (Nguyen et al., 2011), in SlickFlow, the controller computes the primary and the disjoint backup paths and then both are encoded in the packet header along with an *alternative* bit, which indicates the current path being used. When the primary path is affected by link failure, a switch will forward the packets through the backup path and change the value of the alternative, which is necessary for the neighbor switch to follow the backup as well. Due to the header space limitations, the alternative path should not exceed 16 hops. Yang et al. (2015) produced a multipath protection scheme for the SDN data center. The study adopted the principle of the disjoint path to improve the network reliability in addition to extending the OpenFlow protocol by adding the *Type* feature to the flow table entries. The controller computes $1 + n$ disjoint paths for each source-destination node, the value of n can be set to match the importance of the service (the higher the value of n , the more important the service is). The *Type* field is used as an indicator to distinguish between the primary and backup paths. When link failures occur, the influenced paths will be identified by the controller and it will proceed to remove the flow entries of the primary (affected) paths from the only source and destination nodes. Thus, the incoming data packets will follow the rules of the remaining backups.

Bianchi et al. (2014) developed *OpenState* as a super set extension of OpenFlow that aims to offload some of the control logic from the network controller to switches. In other words, the pragmatic approach of OpenState has moved the data plane elements from stateless to stateful operation in network switches. This means that the OpenFlow switches can also be directly programmed and therefore they are not fully dump and are able to handle forwarding rules locally without need for the intervention of the controller. Despite the OpenState has not been adopted yet as a standard extension to OpenFlow, it is expected to be implemented in the future version of OpenFlow (Borokhovich et al., 2018) since the new capabilities of OpenState encourage researchers to further investigate the dependability of the data plane in SDNs. In this context, (Capone et al., 2015) introduced a new OpenState-based protection method for single node and link failures with a view to consider some QoS objectives (e.g. link congestion). The follow-up work SPIDER (Cascone et al., 2016),(Cascone et al., 2017) from the same research group proposed a new method to enable the switches to customize the fault detection and traffic rerouting by relying on OpenState, however, this approach is still inapplicable as the existing SDN equipment does not support such customization.

Van Adrichem et al. (2016) produced a method for pre-planning all the possible backup paths that are necessary to recover from all the expected future failures including both links and nodes. The authors also show how to implement a fast failure detection mechanism based on the liveness monitoring protocols.

Thorat et al. (2016) introduced a new protection scheme in which the VLAN tagging technique was used to aggregate the disrupted flow. The authors show how their proposed method reduced the number of flow rules and accelerates the process of recovery, which fully satisfy the carrier-grade recovery requirements. This method was extended in (Thorat et al., 2017), where the authors considered two methods to enhance their previous work in terms of fast failure recovery. Firstly, the authors proposed the Local Immediate (LIm) recovery strategy, in which the controller will utilise the fast failover to install the alternative paths and the VLAN tagging feature to tag the arriving packets with the outgoing link ID that should be used for the transmission. When a link failure occurs, the switch will automatically divert the disrupted flows to the output port of the backup path and, therefore, this strategy eliminated installing rules per flow and instead, it elevated the flow-aggregation for all flows who share the same output links. Secondly, the authors produced the immediate controller dependent (ICoD) recovery mechanism in which a controller intervention is required. With ICoD, the controller needs to compute the alternative path at the moment of failure and update the failover group table, however, the new installed rules will be dedicated to the affected aggregated-flow and, therefore, this scheme eliminates the number of required rules (i.e. rules for flow-aggregation instead of per flow). Aggregating flows might be a good idea for tackling the failure issue, however, in such a case all the affected flows will be treated as one flow, in other words, different QoS objectives can not be applied. ICoD is similar in some ways to the proposed Algorithm (Node-to-Node) in Chapter 5.

In general, the protection solutions require additional information, which has to be loaded into the data plane elements, in order to tell the nodes how to perform when failure occurs. However, the extra loaded information affects the storage memory of the network switches and thus the designed fault tolerance mechanisms should consider the limited space of the flow table and TCAM.

Since using protection mechanisms are memory (TCAM) consuming due to the requirement of installing backup paths along with the primary ones, this issue has been considered by some researchers, but there are a very limited number of studies in this area. Mohan et al. (2017) produced a protection scheme, as an extension to their previous work presented in (Mohan et al., 2015), that minimises TCAM consumption. The authors developed two routing strategies: *Backward Local Rerouting* (BLR) and *Forward Local Rerouting* (FLR). In BLR, a node-disjoint path is computed as a backup for every primary path in the network and when a failure occurs, packets are sent back to the origin node to be rerouted over the pre-planned backup towards the destination. In FLR, a backup route for each link in the primary path is pre-computed. When a link failure occurs, the packets will be forwarded from the point of failure towards the downstream switch in the primary path by following the backup path, however, in the case that there are multiple backups, the one with the least number of switches will be chosen. Instead of using fast failover group type, the authors have extended the OpenFlow protocol by adding an additional entry, namely: `BACKUP_OUTPUT` to the `ACTION SET` of the flow table entries, so that the new added entry is responsible for setting the output port when a link fails.

Stephens et al. (2016) proposed a new flow tables compression algorithm, as well as a compression-aware routing concept to enhance the ratio of the gained compression rate. The proposed algorithm reduces the consumed TCAM space by using the wildcard to match the tables who share the same output and packet modification operations and hence the compression. The authors relied on their previous work (Stephens et al., 2013) in which they proposed Plinko as a new forwarding model where the forwarding table entries apply the same action.

Zhang et al. (2017a) discussed the problem of the protection schemes and their impact on the shortage of TCAM memory. The authors proposed the *Flow Entry Sharing Protection* (FESP), which is a greedy algorithm that selects the node with the largest available flow entry capacity and the minimum backup flow entry. The study showed how the total number of flow entries can be minimised and the experimental results revealed that the reduction ratio of flow entries is up to 28.31% compared with the existing path and segment protection methods.

B) Hybrid solutions

Some of the legacy network strategies have been utilised to leverage the SDN capability that relates to quick reaction to failures. In this context, Tilmans and Vissicchio (2014) introduced a new architecture called IGP-as-a-Backup SDN (*IBSDN*) in which the centralised network controller is responsible for

pushing the fine-grained primary paths, however, the IGP protocol will provide the alternative paths in case of link failures. Authors showed how an IGP-support local routing agent can be run on top of the OpenFlow switches and can construct and save the paths information independently from the network controller. Watanabe et al. (2015) introduced *ResilientFlow* as a distributed failure mechanism that enables the SDN switches to maintain the link failure that occurs between the controller and switch. *ResilientFlow* runs the OSPF protocol on top of SDN nodes in order to collect and exchange the network information so that the switch will be able to find an alternative path to connect to the controller after losing its connection. The comprehensive studies of (Sinha and Haribabu, 2017) and (Amin et al., 2018) analyse and summarise the existing approaches of the hybrid SDN-Legacy networks with a taxonomy to identify the gaps, limitations and cutting edges in the body of this research.

Restoration

After the failure is detected, the restoration mechanism will be activated. Since the infrastructure layer equipment in SDN are dummy forwarding elements due to the split architecture, then, the central controller is responsible for calculating the alternative paths and then installing the flow entries (i.e. forwarding rules) in the relevant switches of each backup.

A) Native solutions

Sharma et al. (2011) and Staessens et al. (2011) discussed the possibility of achieving the carrier-grade reliability requirement in terms of recovery from failure within 50 μ s, or less. The strategy followed the LoS to detect the changes (such failure) in the network topology, when the controller receives a notification about link failure, the affected paths will be identified by the controller as a first step before recalculating the new alternative paths and sending the respective flow modifications to switches. In both studies, the experiments, which have been carried out on small-scale network (i.e. 6 and 14 nodes), showed that the carrier-grade reliability requirement can be accomplished (i.e. recovery time < 50 μ s). However, the authors stated that the restoration time also depends on the number of flows (i.e. rules) that need to be modified in the affected path, therefore, the recovery time reached up to 300 μ s in some of their experiments that were carried out on 14 nodes and hence it is a significant challenge to meet the reliability requirement of carrier-grade in large-scale networks.

Kuźniar et al. (2013) proposed a restoration scheme called *Automatic Failure Recovery for OpenFlow* (AFRO), which is working on the basis of two phases namely: *record mode* and *recovery mode*. In record mode, AFRO records all the controller-switches activities (e.g. PACKET-IN and FLOW-MOD) towards creating a clean state copy about the network when no failure is experienced. When failure occurs, AFRO switches to the recovery mode followed by generating a new instance of controller (called *shadow controller*), which is a copy from the original state but excluding the failed elements. Then, all the recorded events will be replicated for the sake of installing the difference rule set between the original and shadow copies. The authors produced a prototype of AFRO, however, the study lacks any simulation results and/or measurements to show the effectiveness of AFRO.

Philip and Gourhant (2014) discussed the convergence delay after failure events and attributed it to the network resiliency. The study showed that the speed of failover is determined by two factors: (i) the distance between the failure site and placement of the controller and (ii) the length of the alternative path that needs to be installed by the controller. To accelerate the failure recovery and maximise the resilience and scalability of the network, the authors proposed multiple centralised controllers to enable the dynamic computation of end-to-end paths.

Kim et al. (2012) produced a SDN fault tolerant system, which is called CORONET, to achieve a fast network recovery from multiple link failures of the data plane. CORONET slices the network into set of VLANs in which the ports of physical switches are mapped to different logical VLAN IDs. The route planning module of CORONET computes multiple link-disjoint paths by using the Dijkstra shortest path algorithm (Dijkstra, 1959). The set of computed paths can be used by the network controller to assign the alternative route when the primary path is functioning incorrectly due to a link failure (single or multiple). In addition, the calculated dis-joint paths can be utilised for load balancing purposes (e.g. change between paths in round-robin fashion) and therefore, the system provides a traffic monitoring module for analysis and dynamic load balancing. The CORONET prototype has been built on top of a NOX controller and it is compatible with the standard OpenFlow protocol.

Jinyao et al. (2015) proposed HiQoS, a SDN-based solution that finds multiple paths between source and destination nodes to guarantee certain QoS constraints such as bandwidth, delay and throughput. HiQoS uses a modified Dijkstra algorithm to gain the multiple paths set that meets the QoS requirements.

Not only can the QoS be guaranteed with HiQoS, but also a fast failure recovery can be achieved. This is because when a link fails it causes a truncation or interruption in some paths, the controller will then directly select a working path from the already computed ones. The authors compared the performance of HiQoS, which supports multiple paths, against MiQoS, which is a single path solution, and the experiments showed that HiQoS outperformed the MiQoS in terms of performance and resilience to failure.

Astaneh and Heydari (2016) considered the problem of the required number of flow entries that are necessary for the restoration operation, which is an extension to their previous work reported in (Astaneh and Heydari, 2015). The authors imputed the required time for the process of restoration and recovery from multiple data plane failures to the number of flow entries that need to be pushed by the network controller, in other words, the more forwarding rules to be established the more delay is incurred and the greater the impact upon the operation of failure recovery. Comparing with the current restoration methods, Astaneh and Heydari argued that fast recovery from link failure cannot be guaranteed by merely relying on shortest path techniques, since it does not consider the pre-installed rules and hence new techniques have to be devised. In this context, the authors introduce optimal and sub-optimal solutions to identify routes by focusing on two metrics, namely, the lowest operation cost and the least path cost. The optimal solution, which is a Dijkstra-like solution, seeks to obtain candidates with minimum path cost, however, the sub-optimal solution endeavours to explore solutions with minimum operation cost in terms of either Add-Only flow entries or Add-and-Remove. The simulation results that have been conducted on two topologies showed that up to 50% of operation cost reduction can be achieved in case of Add-and-Remove, however, the reduction ratio of the operation cost can be reached up to 30% in the case of Add-Only. The study makes no attempt to address the question of how the reduction in the operation cost will not affect the order of the established rules of the new solution, in other words, it does not guarantee a sequential chain for the alternative path. The proper order of nodes in a path is important and selecting an alternative path on the basis of mere presence of common nodes may not always work. Some cross-sectional studies consider the correlation between reducing the flow entries and the original sequence of rules of paths while computing the alternatives. Malik et al. (2017a) developed a couple of new algorithms that divide the anatomy of a path to achieve a quick and optimum solution to the problem of finding and replacing a failed link. The authors demonstrated how the proposed algorithms utilise existing pre-installed flow entries through dealing with a sub-path of the original shortest path at the moment of failure. It has been also noted that the resulting new path, though not necessarily a shortest path itself, is guaranteed to have better utilisation of the existing rules than in the case of an end-to-end path discovery algorithm, therefore leading to the acceleration of the operation of path recovery in terms of both minimising the discovery time and the number of flow entry modifications. In the same vein, Malik et al. (2017c) suggested a new approach for an efficient restoration after a link failure. On one hand, the authors proposed a method to improving the performance of reactive failure recovery through segmenting the network topology into a certain number of non-overlapping cliques. By dividing the network into N cliques, the authors considered the only intra link failure by assuming that when the link failure occurs then only one clique will suffer from the failure. Meantime, the other cliques should be working fine. Therefore, the only clique which includes the effected link will be treated rather than dealing with the whole network topology. On the other hand, the study has not dealt with inter link failure that affects the connection between the cliques itself.

Since the studies that considered the reduction of flow entries were mainly rely on the possibility of deriving a new least cost route out of the affected one. Therefore, Malik et al. (2017b) presented a study that dealt with the problem of path selection. The authors proposed an algorithm that enables the selection of the best shortest path, which is essential to guarantee the feasibility of path that required a minimal number of flow entries to mask the failure when occurs, towards fast reconfiguring data plane.

B) Hybrid solutions

Yu et al. (2011) proposed a framework for tackling the slow recovery of OSPF against failures. The authors employed an OpenFlow controller to reactively manage the data plane failures by dynamically recalculating paths using the Floyd algorithm (Floyd, 1962) and then, installed the new computed path rules to mask the occurred failure. However, during the normal operation, the OpenFlow-enabled switches rely on a *Routing Information Database* (RIB) to handle the incoming data packets.

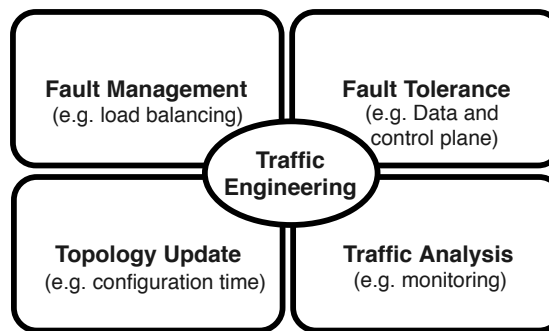


Figure 9. Traffic engineering scope in SDNs

622

623 C) *Traffic engineering based solutions*

624 Traffic engineering is becoming an increasingly essential feature for network administrators, which
 625 strives to maximise the network resource utilisation and optimise the network performance towards
 626 delivering a good service to end users. One of the key roles of traffic engineering is routing, and the key
 627 challenge for routing is to find good routes that attain a desired level of network performance (i.e. QoS
 628 and SLA) and to adapt to network events that lead to continuous changes in network topology such as the
 629 frequent incidents of link and node failures (Wang et al., 2008). Traditionally, the distributed protocols
 630 (e.g. BGP and OSPF) are used to enable the adjacent nodes from sharing their control information
 631 due to the lack of global view available in traditional networks (Hakiri et al., 2014). In contrast, and
 632 particularly with the global view supported by the controller, the traffic engineering in SDN becomes
 633 more efficient and intelligent where such an improvement meant that traffic engineering is involved in
 634 four domains, which includes; fault management, fault tolerance, topology update and traffic analysis
 635 (Akyildiz et al., 2014) as illustrated in Figure 9. The recent studies in (Akyildiz et al., 2016; Mendiola
 636 et al., 2016) revealed that fault tolerance and recovery from failures is one of the key challenges that
 637 face SDN nowadays. In this regard, Luo et al. (2015) introduced the ADaptive Multi-Path Computation
 638 Framework (ADMPCF) as a traffic engineering tool for large scale OpenFlow network systems. ADMPCF
 639 can hold two, or more, disjoint paths to be utilised at the moment of occurrence of some network events
 640 (e.g. link failure, congestion, etc.). As a result, those precomputed paths can be used as backups in
 641 the case of link/node failures or when the defined cost function does not meet the QoS requirements.
 642 Similarly, Dinh et al. (2016) proposed a multipath traffic engineering approach for SDN (MSDN-TE) in
 643 which k-paths between the source and destination pairs will be computed and established. On one hand,
 644 this approach is employed to reduce the load on congested areas by selecting the least loaded path for
 645 handling the new arriving data flows. On the other hand, MSDN-TE can also be used as a technique to
 646 improve the network resilience against failures (e.g. link failure), since the multiple established paths
 647 between pairs will allow the service to be available even if one of the pre-installed paths fails.

648 **Protection-Restoration Combination**

649 Some studies suggested cooperation between the two failure recovery schemes (i.e. protection and
 650 restoration) to seek a better approach to enhance the fault management of SDN. Sahri and Okamura
 651 (2014) proposed a fast failure recovery scheme that considers the connectivity and optimality factors.
 652 According to their approach, the controller calculates the backup paths in advance and when a failure
 653 occurs the respective switch will firstly, redirect the affected flow to the backup path and secondly, notify
 654 the controller in order to compute an optimal path.

655 Thorat et al. (2015) proposed the Optimized Self-Healing (OSH) framework for optimising the process
 656 of failure recovery. With OSH, the failure recovery can be classified into two steps: (i) Rapid Recovery
 657 (RR) by relying on the fast failover group type mechanism, in which the recovery process is performed
 658 locally without requiring intervention by the controller and afterward (ii) the network controller will
 659 optimise the recovered paths towards meeting the prescribed QoS level.

660 In the same way, Wang et al. (2018) have formalised the recovery process as a multiobjective optimi-
 661 sation problem, in order to gain a fine-grained trade-off between recovery time and QoS requirements,
 662 such as resource utilisation. The authors also devised the Cooperative Link Failure Recovery scheme

(CFR) to ameliorate the process of failure recovery in SDNs. CFR consists of two algorithms, namely *rapid connectivity restoration* and *backup path adjustment*. The rapid connectivity restoration algorithm aims to recover from failure quickly through calculating the backup paths for all source-destination pairs and establishing those paths in the data plane switches. The fast failover mechanism has been utilised to detour disrupted flows to the pre-computed backup paths, in addition, VLAN tags technique to aggregate affected flows was used to reduce the installed backup rules. The backup path adjustment algorithm aims to refine the paths that are set by the rapid connectivity restoration, which is important for QoS.

3.3 Factors Impacting Recovery Process

The recovery process delay is connected with some aspects that can be categorised into three groups with each representing a different stage in the recovery process cycle. For instance, some factors are related to the controller side as it is the most vital part in the network, while, some other factors are associated with the OpenFlow protocol specifications, as it is the standard communication interface between the data and control layers. There are also some other relevant and important factors such as those resulting from the data plane nodes performance. The following three subsections will cover the aforementioned aspects.

3.3.1 Network Changes Discovery

The first step to recover from a link failure is to discover the changes in the underlying network topology. In SDNs, the topology changes can be detected on the basis of the infrastructure components, which are of two classes, nodes and links. On one hand, the liveness of nodes (e.g. switches) can be verified by sending an ECHO message regularly from the control plane and, therefore, it is not difficult for the controller to detect the disconnected switches. On the other hand, the operational state of links can be checked by the switches they are connected with. For this purpose, two mechanisms can be utilised to perform a link failure detection, namely BFD and LoS (as mentioned in Section 3.1). OpenFlow protocol supports the LoS mechanism through its PORT-STATUS messages that provide the controller with notifications as soon as a port change status is detected. A number of studies took this matter into account due to its critical impact on failure recovery.

Desai and Nandagopal (2010) discussed the problem of forwarding data traffic towards a non-operational area that is experiencing a link failure in a centralised controlling network such as an SDN. The authors claimed that the period of time from the moment of failure till the moment of recovery will usually cause a waste of bandwidth due to the fact that switches will continue forwarding flows, regardless of the failure, that are unnecessarily transmitted until an update arrives from controller. This waste is substantial and must avoid. A controller could be placed remotely and may not be connected directly to the switches that belong to its domain, which makes the update operation even more critical and time consuming. For this reason, the study focuses on restraining switches from sending traffic in the direction where the failed link is located and this should not require the controller intervention in order to speed up the process. To address this, the Link Failure Messages (LFMs) algorithm was developed, which enables the sharing of failure information among the relevant switches and therefore it prevents them from sending data packets to that direction. LFMs does not require the controller intervention and hence the relevant switches will be aware of the link failures quickly without waiting for controller messages. However, the performance of LFMs is related to the number of involved switches in the process, i.e., the large the number of involved switches the longer the time it takes to send LFM to all them.

Gebre-Amlak et al. (2017) developed the Topology-Aware Reliability Management framework (TAR-Man) that classifies the network elements into different zones based on their criticality, the more critical the more important. Instead of sending LLDP messages at uniform intervals from the controller to switches, TARMan customises the frequency of LLDPs according to how the zone is important (the more important the more frequent LLDP). Therefore, TARMan aims to reduce the time of failure detection for the critical nodes/links in the network.

3.3.2 Path Computation

Generally, and for restoration schemes, the step of path computation will be activated as soon the first step (i.e. failure detection) is done. When failure is detected an alternative working path should be computed reactively, unless it is planned proactively. The OpenFlow switches are not able to calculate new paths when failure occurs owing to the SDN architecture and the decoupling between the data and control plane, however, controller is able to do so utilising the global view of the network's entire resources. Currently, most of the reactive failure recovery techniques suspend the alternative path computation

until failure occurs, except some traffic engineering methods such as the ADMPCF (Luo et al., 2015), which continuously strive to look for good paths in the network that will be stored as backups and to be employed when an incident occurs. Similarly, Mendiola et al. (2015) developed the Dynamic Path Computation (DynPac) framework as a traffic engineering technique to improve the network resource utilisation. DynPac uses a stateful path computation element (PCE) that computes and stores all possible paths between all source and destination pairs in a database and it guarantees the bandwidth allocation for the requested services. Beside this, the authors state that DynPac is not only concerned with resource utilisation improvement, but also it supports resilience during link failures through removing the flow entries of the failed path and installing the flow entries of the path that will be selected from the database of paths.

The phase of path computation also constitutes an obstacle to the process of recovery as this will lead the controller to put in additional time to search for possible solutions, while, the controller is racing against time to mitigate the effect of disruption and speed-up the recovery operation. A further discussion on this issue can be found in (Malik et al., 2017a).

3.3.3 Network Update

After the second stage (i.e. path computation) is performed, the next and last phase will be commenced, these are to update the relevant switches with the new computed flow entries that will mask the disruption caused by failure incidents (e.g. link failure). This can be done either by pushing a new set of flow entries associated with a higher priority than the idle ones, as a technique to override the existing rules, or by removing the non-working rules along the damaged path first and installing the new forwarding entries. Network update is not trivial and sometimes it leads to a policy conflict and/or unwanted loop. The SDN controller is responsible for updating the switches whenever a reconfiguration is needed. To do so, the controller uses some of the OpenFlow protocol messages (i.e. such as those mentioned in Section 2.2) to reconfigure the switches' flow table according to the forwarding rules of the new configuration scheme. Since the network is prone to planned and unplanned events as mentioned in Section 3, this means the network updates and reconfiguration can be divided into two categories, as follows:

a) Planned reconfiguration

Changes are not only demanded during the unplanned failure incidents, but also for some planned maintenance activities. Reitblatt et al. (2011, 2012) discussed the problem of consistent update and transition from an old to a new configuration for the planned occasions. The authors propose two strategies: *per-packet* and *per-flow* for a consistent update. The per-packet strategy duplicates the table entries for switches so that the controller needs to amend the flow tables by installing the new flow entries that meet the new configuration.

The controller then conducts the necessary update for the ingress switches and since they cannot be updated at the same time and, therefore, it is necessary to mark the incoming packets (e.g. using VLAN tags) with the version type of the configuration that needs to be followed. In such a case, the incoming packets will be processed either according to the configuration prior to the update "old" or in line with the recent update "new", but not a mixture of the two. When all the incoming packets start following the new configuration, the controller will remove the flow entries of the old configuration.

Similarly, the per-flow mechanism behaves as per-flow, however, it ensures that all packets that belong to the same flow will be handled with the same policy (i.e. old or new). This is because in some cases it is compulsory for the packets belonging to the same flow to reach a certain destination, for instance, the server load-balancer application necessitates all packets of the same TCP connection to go through the same server replica.

The implementation of per-flow is more complex than per-packet since the system needs to be aware of the active packets for each flow. Therefore, for per-flow, the controller needs to push the new flow entries of the new configuration into switches. Subsequently, on the ingress switches, the controller sets a timeout for the old configuration rules and pushes the new configuration rules with low priority. Hence, as soon as the flows match the old rules finish, it will expire and the rules of the new configuration will take effect. Planned failures comprise 20% of all failures, which is mainly due to the scheduled network maintenance activities (Markopoulou et al., 2008).

b) Unplanned reconfiguration

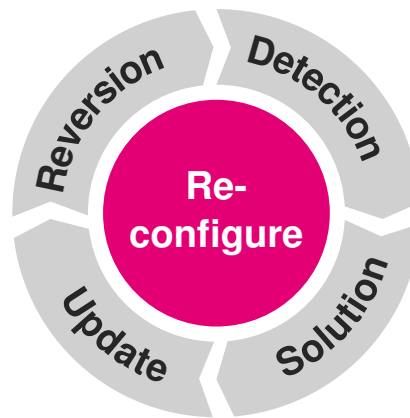


Figure 10. Reconfiguration cycle in SDNs

Unplanned reconfiguration is the necessary changes that need to be carried out in order to keep the network functioning properly in case one, or more, of its components stop working and hence the fault tolerance property is important. Time plays a significant role in the process of reconfiguration as it is axiomatic that network operators are willing to provide their services to the end-users continuously and with no deterioration. Therefore, the less reconfiguration time is consumed, the more preferable the technique.

The performance of current OpenFlow switches varies according to their vendor, it was reported that the range of time required for inserting a single flow entry varies from $0.5 \mu s$ to $10 \mu s$ (Rotsos et al., 2012; Tran-Gia, 2018). Therefore, the required time for rule insertion is not negligible and can not be ignored, especially from the point of view of a path failure, since at the moment of failure, we are not only concerned with new flow rule insertion, but also with the removal of invalid old rules. Jin et al. (2014) demonstrated that a time of $11 \mu s$ (at least) is required for each rule modification; i.e. the latency caused by the insertion and deletion operations. To accelerate operation of changes, Mizrahi and Moses (2016) suggested a simultaneous update approach to replace flow entries simultaneously as a step to reduce the packet loss rate while reconfiguring the network paths. However, Vissicchio et al. (2017) considered such simultaneous operations as impractical since it is impossible to predict the rules installation delay, in other words, it cannot be guaranteed that all the respective switches will proceed and complete the updates at the same time and hence it is unsafe and potentially highly likely to cause forwarding and policy violations. In contrast, the ordered update (i.e. node by node) preserves the correctness of the forwarding policy, but it takes a longer time compared with simultaneous update.

Undoubtedly, switches must be reconfigured individually as it is very hard to perform the network update with one atomic step (Reitblatt et al., 2011), which means the update time is tied to the number of switches involved in the process of reconfiguration, in other words, the larger the number of switches the more time is required. Typically, the quicker update operation accelerates the recovery process and hence, the less the services are disrupted by network failures. An experimental investigation is presented in Chapter 4, which discusses this issue in the context of path recovery from link failures.

4 SUMMARY OF LIMITATIONS

Resilience to failures and convergence time after a failure are significant factors that impinge on the network performance. Despite the advantages of SDN and the dramatic network improvements it brought, this innovation has been accompanied by several challenges, such as the management of network failures, the reconfiguration of the network architecture and the monitoring of its status. These multiple issues are connected to one root subject, which is the network reconfiguration. The SDN reconfiguration cycle is illustrated in Figure 10 in which the reconfiguration process has been divided into four main parts, namely *Detection*, *Solution*, *Update* and *Reversion*. Each part stresses the elasticity of the reconfiguration cycle differently. However, the issues involved with these parts and considered in this thesis can be summarised as follows:

- *TCAM space limitation*: Since pushing extra flow entries as backups increases the chance of

overwhelming TCAM memory, which will lead to inefficient use of SDN resources (i.e. forwarding elements like switches). Moreover, the installation of many attributes will affect the process of match and action of the data plane forwarding elements. Furthermore, there is no guarantee that the pushed flow entries as preplanned paths are failure-free, in other words, the backup paths could fail earlier than the primary ones, which could further burden the controller with unnecessary tasks. Consequently, protection methods are not the best solution to tackle failures especially for the TCAM-limited SDNs. Therefore, this thesis advocates for the restoration strategies.

- *Alternative calculation:* Reactive-based solutions usually calculate the secondary (i.e. backup) paths at the moment of detecting failures. This calculation process is not trivial if the network operator is biased towards adhering to the carrier-grade quality. This issue was considered by some of the literature (as mentioned in Section 3.3.2). Interestingly, some publications have suggested the disjointness as a constraint for the calculated paths, however, this will increase time of update in a proportional manner to the number of rules that need to be changed.
- *Restore delay:* Given that SDN is a centralised networking system in which the network controller is responsible for updating the data plane forwarding components and since the update operation takes time as it cannot be established in one go, therefore, failure recovery schemes with a minimum number of updates are more preferable. Proactive schemes are less affected by this matter than the reactive methods, since in protection both primary and secondary paths are installed, while this is not the case for the restoration.
- *Reverting after repair:* Changing paths according to failure incidents is not enough as this will leave the network in a not fully-optimal state. Since the paths resulting from the recovery operation could be less ideal than before the failure (e.g. the alternative path is longer than the original), therefore, it should be an interim solution until the affected primary path gets back to the operational state again. This reversion is important and required to keep the network operating efficiently in an optimal state. This part of reversion is missed from the literature of the reactive schemes and hence it should be emphasised and discussed explicitly.

Although the efficiency of failure recovery mechanisms are partially restricted by the time of failure detection, this thesis has no contribution to enhance this part of the reconfiguration cycle. In general, proactive failure recovery schemes are fast in terms of reacting to failure (i.e. small delay), however, it requires a large number of preplanned routing rules, which consumes the storage of data plan devices (TCAM space) and most of the studies in this area have been tested on small-scale networks (e.g. between 7-28 nodes and 7-43 links), hence they are not directly comparable to a large-scale network. Furthermore, some of the protection schemes utilised some Openflow specifications such as VLANs to perform light weight failure recovery mechanisms, but, in such cases the VLANs functionality is deactivated and cannot be used any more. However, some of the protection proposals are still not supported (e.g. OpenState) by the current OpenFlow, making such a contribution impractical in reality.

In contrast, the reactive failure recovery schemes are not as fast as proactive ones, but, no extra information on routing tables is required and so, it does not exhaust the memory space of the data plane elements.

5 OPEN RESEARCH CHALLENGES

This paper suggests several future directions that researchers can undertake to meet the requirement of SDN fault tolerance and therefore the SDN dependability. Since the current research focused on either protection or restoration mechanisms to mask link failures, some certain scalability drawbacks are associated with both mechanisms as follows:

- *Memory:* memory space consumption is the main key challenge that impedes protection techniques. This issue becomes particularly critical for large scale networks as huge number of backup flow entries need to be installed and therefore memory space is needed to store such a huge number of forwarding rules. Based on the literature, most of the proposed solutions rely on the OpenFlow implementation schemes such as VLANs, however, such solutions affects the use of VLANs itself. In other words, utilising VLANs for fault tolerance purposes will cancel the actual function of VLANs. Therefore, more investigation is required.

- *Time*: failure recovery with restoration strategy is time consuming because the network controller will need to update the affected routes after detecting a broken connection. This issue becomes more particularly critical for large scale networks as the longer path, the more time to update. Based on the literature, most of the proposed solutions who tackled this issue failed to guarantee the shortest path as an alternative. Therefore, more research is required to investigate whether it is possible to accelerate the recovery process as well as guarantee the alternative shortest path at same time.
- *Policy interaction*: Most of the current solutions only consider the only recovery aspect without taking into account some other important properties such as the security policy. Traditionally, the interaction between fault tolerance and security has been considered (e.g. (Price, 1999)). However, in SDNs this topic still awaits further investigation, for instance, a proposed solution must avoid policy violations by installing new alternative paths that should not bypass security policies.

6 CONCLUSION

This paper surveys several prominent and promising methods that have been proposed to tackle and solve the issue of data plane fault tolerance. The paper first reviewed the past exerted efforts towards favoring programmable networks down to the age of software-defined networking. Dependable systems in terms of communications paradigm is covered with highlights on the fault tolerance attribute and failure recovery schemes in computer networks in general and SDN in particular. Then, the pros and cons of literature on SDN fault tolerance and failure recovery are summarised with an emphasis on the current challenges that hamper the SDNs in the context of performance and resource utilisation. The paper also presented a classification of the various factors that impacting the process of failure recovery for better understanding about the relationship between the existing body of SDN research as well as to identify the cutting edge research issues. Finally, the current issues and challenges are discussed and categorized into four groups with several questions still remain to be answered and therefore more research is necessary to fill the remaining gaps.

Notwithstanding large literature on SDNs, there are still some issues and challenges related to SDN dependability by means of fault tolerance where more investigation is needed to address the open challenges of scalability and resilience.

REFERENCES

- Abdelaziz, A., Fong, A. T., Gani, A., Garba, U., Khan, S., Akhunzada, A., Talebian, H., and Choo, K.-K. R. (2017). Distributed controller clustering in software defined networks. *PloS one*, 12(4):e0174715.
- Akyildiz, I. F., Lee, A., Wang, P., Luo, M., and Chou, W. (2014). A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30.
- Akyildiz, I. F., Lee, A., Wang, P., Luo, M., and Chou, W. (2016). Research challenges for traffic engineering in software defined networks. *IEEE Network*, 30(3):52–58.
- Amin, R., Reisslein, M., and Shah, N. (2018). Hybrid sdn networks: A survey of existing approaches. *IEEE Communications Surveys & Tutorials*.
- Astaneh, S. and Heydari, S. S. (2015). Multi-failure restoration with minimal flow operations in software defined networks. In *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the*, pages 263–266. IEEE.
- Astaneh, S. A. and 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.
- Benson, T., Akella, A., and Maltz, D. A. (2009). Unraveling the complexity of network management. In *NSDI*, pages 335–348.
- Bhattacharjee, S., Calvert, K. L., and Zegura, E. W. (1997). An architecture for active networking. In *High Performance Networking VII*, pages 265–279. Springer.
- Bianchi, G., Bonola, M., Capone, A., and Cascone, C. (2014). Openstate: programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM Computer Communication Review*, 44(2):44–51.
- Borokhovich, M., Rault, C., Schiff, L., and Schmid, S. (2018). The show must go on: Fundamental data plane connectivity services for dependable sdn. *Computer Communications*, 116:172–183.

- 908 Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. (2005). Design and
909 implementation of a routing control platform. In *Proceedings of the 2Nd Conference on Symposium on*
910 *Networked Systems Design & Implementation-Volume 2*, pages 15–28. USENIX Association.
- 911 Cai, Z., Cox, A. L., and Ng, T. (2010). Maestro: A system for scalable openflow control. Technical report.
- 912 Capone, A., Cascone, C., Nguyen, A. Q., and Sanso, B. (2015). Detour planning for fast and reliable
913 failure recovery in sdn with openstate. In *Design of Reliable Communication Networks (DRCN), 2015*
914 *11th International Conference on the*, pages 25–32. IEEE.
- 915 Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking
916 control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages
917 1–12. ACM.
- 918 Casado, M., Garfinkel, T., Akella, A., Freedman, M. J., Boneh, D., McKeown, N., and Shenker, S. (2006).
919 Sane: A protection architecture for enterprise networks. In *USENIX Security Symposium*, volume 49,
920 pages 137–151.
- 921 Cascone, C., Pollini, L., Sanvito, D., Capone, A., and Sanso, B. (2016). Spider: Fault resilient sdn pipeline
922 with recovery delay guarantees. In *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, pages
923 296–302. IEEE.
- 924 Cascone, C., Sanvito, D., Pollini, L., Capone, A., and Sansò, B. (2017). Fast failure detection and recovery
925 in sdn with stateful data plane. *International Journal of Network Management*, 27(2).
- 926 Cérin, C., Coti, C., Delort, P., Diaz, F., Gagnaire, M., Gaumer, Q., Guillaume, N., Lous, J., Lubiarz, S.,
927 Raffaelli, J., et al. (2013). Downtime statistics of current cloud solutions. *International Working Group*
928 *on Cloud Computing Resiliency, Tech. Rep.*
- 929 Dantu, R., Anderson, T. A., Gopal, R., and Yang, L. L. (2004). Forwarding and control element separation
930 (forces) framework.
- 931 Desai, M. and Nandagopal, T. (2010). Coping with link failures in centralized control plane architectures.
932 In *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*,
933 pages 1–10. IEEE.
- 934 Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*,
935 1(1):269–271.
- 936 Dinh, K. T., Kukliński, S., Kujawa, W., and Ulaski, M. (2016). Msdn-te: Multipath based traffic
937 engineering for sdn. In *Asian Conference on Intelligent Information and Database Systems*, pages
938 630–639. Springer.
- 939 Erickson, D. (2013). The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM*
940 *workshop on Hot topics in software defined networking*, pages 13–18. ACM.
- 941 Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., and Van Der Merwe, J. (2004). The case for
942 separating routing from routers. In *Proceedings of the ACM SIGCOMM workshop on Future directions*
943 *in network architecture*, pages 5–12. ACM.
- 944 Feamster, N., Rexford, J., and Zegura, E. (2014). The road to sdn: an intellectual history of programmable
945 networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98.
- 946 Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345.
- 947 Gebre-Amlak, H., Banala, G., Song, S., Choi, B.-Y., Choi, T., and Zhu, H. (2017). Tarman: Topology-
948 aware reliability management for softwarized network systems. In *Local and Metropolitan Area*
949 *Networks (LANMAN), 2017 IEEE International Symposium on*, pages 1–6. IEEE.
- 950 Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and
951 Zhang, H. (2005). A clean slate 4d approach to network control and management. *ACM SIGCOMM*
952 *Computer Communication Review*, 35(5):41–54.
- 953 Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). Nox:
954 towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*,
955 38(3):105–110.
- 956 Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D. C., and Gayraud, T. (2014). Software-defined networking:
957 Challenges and research opportunities for future internet. *Computer Networks*, 75:453–471.
- 958 Hedrick, C. L. (1988). Routing information protocol. Technical report.
- 959 Jin, X., Liu, H. H., Gandhi, R., Kandula, S., Mahajan, R., Zhang, M., Rexford, J., and Wattenhofer, R.
960 (2014). Dynamic scheduling of network updates. In *ACM SIGCOMM Computer Communication*
961 *Review*, volume 44, pages 539–550. ACM.
- 962 Jinyao, Y., Hailong, Z., Qianjun, S., Bo, L., and Xiao, G. (2015). Hiquos: An sdn-based multipath qos

- 963 solution. *China Communications*, 12(5):123–133.
- 964 Katz, D. and Ward, D. (2010). Bidirectional forwarding detection (bfd). Technical report.
- 965 Kim, H., Schlansker, M., Santos, J. R., Tourrilhes, J., Turner, Y., and Feamster, N. (2012). Coronet: Fault
- 966 tolerance for software defined networks. In *Network Protocols (ICNP), 2012 20th IEEE International*
- 967 *Conference on*, pages 1–2. IEEE.
- 968 Kitsuan, N., McGettrick, S., Slyne, F., Payne, D. B., and Ruffini, M. (2015). Independent transient plane
- 969 design for protection in openflow-based networks. *IEEE/OSA Journal of Optical Communications and*
- 970 *Networking*, 7(4):264–275.
- 971 Kitsuan, N., Payne, D. B., and Ruffini, M. (2014). A novel protection design for openflow-based
- 972 networks. In *Transparent Optical Networks (ICTON), 2014 16th International Conference on*, pages
- 973 1–5. IEEE.
- 974 Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015).
- 975 Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.
- 976 Kurtz, F. and Wietfeld, C. (2017). Advanced controller resiliency in software-defined networking enabled
- 977 critical infrastructure communications. In *Information and Communication Technology Convergence*
- 978 *(ICTC), 2017 International Conference on*, pages 673–678. IEEE.
- 979 Kuźniar, M., Perešini, P., Vasić, N., Canini, M., and Kostić, D. (2013). Automatic failure recovery for
- 980 software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in*
- 981 *software defined networking*, pages 159–160. ACM.
- 982 Lakshman, T., Nandagopal, T., Ramjee, R., Sabnani, K., and Woo, T. (2004). The softrouter architecture.
- 983 In *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, volume 2004. Citeseer.
- 984 Laprie, J., Avizienis, A., and Kopetz, H. (1992). Dependable computing and fault-tolerant systems.
- 985 *Depndability: Basic Concepts and Terminology in English, French, German, Italian and Japanese*, 5.
- 986 Lin, P., Bi, J., Hu, H., Feng, T., and Jiang, X. (2011). A quick survey on selected approaches for preparing
- 987 programmable networks. In *Proceedings of the 7th Asian Internet Engineering Conference*, pages
- 988 160–163. ACM.
- 989 Luo, M., Zeng, Y., Li, J., and Chou, W. (2015). An adaptive multi-path computation framework for
- 990 centrally controlled networks. *Computer Networks*, 83:30–44.
- 991 Malik, A., Aziz, B., Adda, M., and Ke, C.-H. (2017a). Optimisation methods for fast restoration of
- 992 software-defined networks. *IEEE Access*, 5:16111–16123.
- 993 Malik, A., Aziz, B., and Bader-El-Den, M. (2017b). Finding most reliable paths for software defined
- 994 networks. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th*
- 995 *International*, pages 1309–1314. IEEE.
- 996 Malik, A., Aziz, B., Ke, C.-H., Liu, H., and Adda, M. (2017c). Virtual topology partitioning towards
- 997 an efficient failure recovery of software defined networks. In *The 16th International Conference on*
- 998 *Machine Learning and Cybernetics (ICMLC)*. IEEE.
- 999 Markopoulou, A., Iannaccone, G., Bhattacharyya, S., Chuah, C.-N., and Diot, C. (2004). Characterization
- 1000 of failures in an ip backbone. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE*
- 1001 *Computer and Communications Societies*, volume 4, pages 2307–2317. IEEE.
- 1002 Markopoulou, A., Iannaccone, G., Bhattacharyya, S., Chuah, C.-N., Ganjali, Y., and Diot, C. (2008).
- 1003 Characterization of failures in an operational ip backbone network. *IEEE/ACM Transactions on*
- 1004 *Networking (TON)*, 16(4):749–762.
- 1005 McCauley, M. (2012). Pox.
- 1006 McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and
- 1007 Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer*
- 1008 *Communication Review*, 38(2):69–74.
- 1009 Melcherts, H. E. (2017). The internet of everything and beyond. *Human Bond Communication: The Holy*
- 1010 *Grail of Holistic Communication and Immersive Experience*, page 173.
- 1011 Mendiola, A., Astorga, J., Jacob, E., and Higuero, M. (2016). A survey on the contributions of software-
- 1012 defined networking to traffic engineering. *IEEE Communications Surveys & Tutorials*, 19(2):918–953.
- 1013 Mendiola, A., Astorga, J., Jacob, E., Higuero, M., Urtasun, A., and Fuentes, V. (2015). Dynpac: A path
- 1014 computation framework for sdn. In *Software Defined Networks (EWSDN), 2015 Fourth European*
- 1015 *Workshop on*, pages 119–120. IEEE.
- 1016 Mizrahi, T. and Moses, Y. (2016). Software defined networks: It’s about time. In *INFOCOM 2016-The*
- 1017 *35th Annual IEEE International Conference on Computer Communications*, IEEE, pages 1–9. IEEE.

- 1018 Mohan, P. M., Truong-Huu, T., and Gurusamy, M. (2015). Tcam-aware local rerouting for fast and efficient
1019 failure recovery in software defined networks. In *Global Communications Conference (GLOBECOM),*
1020 *2015 IEEE*, pages 1–6. IEEE.
- 1021 Mohan, P. M., Truong-Huu, T., and Gurusamy, M. (2017). Fault tolerance in tcam-limited software
1022 defined networks. *Computer Networks*, 116:47–62.
- 1023 Mourtzis, D., Vlachou, E., and Milas, N. (2016). Industrial big data as a result of iot adoption in
1024 manufacturing. *Procedia CIRP*, 55:290–295.
- 1025 Moy, J. (1998). Ospf version 2. Technical report.
- 1026 Nguyen, G. T., Agarwal, R., Liu, J., Caesar, M., Godfrey, P., and Shenker, S. (2011). Slick packets. *ACM*
1027 *SIGMETRICS Performance Evaluation Review*, 39(1):205–216.
- 1028 Nunes, B. A. A., Mendonca, M., Nguyen, X.-N., Obraczka, K., and Turetti, T. (2014). A survey of
1029 software-defined networking: Past, present, and future of programmable networks. *IEEE Communica-*
1030 *tions Surveys & Tutorials*, 16(3):1617–1634.
- 1031 ONF (2011). Openflow switch specification: Version 1.1.0 implemented (wire pro tocol 0x02). Technical
1032 report.
- 1033 ONF (2018). Open networking foundation.
- 1034 ONF Market Education Committee (2012). Software-defined networking: The new norm for networks.
1035 *ONF White Paper*.
- 1036 OpenDaylight (2013). A linux foundation collaborative project.
- 1037 Peter, A. L. and Thomas, A. (1981). *Fault Tolerance: Principles and Practice*. Springer-Verlag Wien
1038 New York, second edition.
- 1039 Philip, V. D. and Gourhant, Y. (2014). Cross-control: A scalable multi-topology fault restoration
1040 mechanism using logically centralized controllers. In *High Performance Switching and Routing*
1041 *(HPSR), 2014 IEEE 15th International Conference on*, pages 57–63. IEEE.
- 1042 Price, G. (1999). The interaction between fault tolerance and security. Technical report, University of
1043 Cambridge, Computer Laboratory.
- 1044 Project Floodlight (2012). Open source software for building software-defined networks.
- 1045 Ramos, R. M., Martinello, M., and Rothenberg, C. E. (2013). Slickflow: Resilient source routing in
1046 data center networks unlocked by openflow. In *Local Computer Networks (LCN), 2013 IEEE 38th*
1047 *Conference on*, pages 606–613. IEEE.
- 1048 Reitblatt, M., Foster, N., Rexford, J., Schlesinger, C., and Walker, D. (2012). Abstractions for network
1049 update. *ACM SIGCOMM Computer Communication Review*, 42(4):323–334.
- 1050 Reitblatt, M., Foster, N., Rexford, J., and Walker, D. (2011). Consistent updates for software-defined
1051 networks: Change you can believe in! In *Proceedings of the 10th ACM Workshop on Hot Topics in*
1052 *Networks*, page 7. ACM.
- 1053 Rotsos, C., Sarrar, N., Uhlig, S., Sherwood, R., and Moore, A. W. (2012). Oflops: An open framework for
1054 openflow switch evaluation. In *International Conference on Passive and Active Network Measurement*,
1055 pages 85–95. Springer.
- 1056 Sahri, N. and Okamura, K. (2014). Fast failover mechanism for software defined networking: Openflow
1057 based. In *Proceedings of The Ninth International Conference on Future Internet Technologies*, page 16.
1058 ACM.
- 1059 Sgambelluri, A., Giorgetti, A., Cugini, F., Paolucci, F., and Castoldi, P. (2013a). Effective flow protection
1060 in open-flow rings. In *National Fiber Optic Engineers Conference*, pages JTh2A–01. Optical Society
1061 of America.
- 1062 Sgambelluri, A., Giorgetti, A., Cugini, F., Paolucci, F., and Castoldi, P. (2013b). Openflow-based segment
1063 protection in ethernet networks. *Journal of Optical Communications and Networking*, 5(9):1066–1075.
- 1064 Sharma, S., Staessens, D., Colle, D., Pickavet, M., and Demeester, P. (2011). Enabling fast failure
1065 recovery in openflow networks. In *Design of Reliable Communication Networks (DRCN), 2011 8th*
1066 *International Workshop on the*, pages 164–171. IEEE.
- 1067 Sharma, S., Staessens, D., Colle, D., Pickavet, M., and Demeester, P. (2013). Openflow: Meeting
1068 carrier-grade recovery requirements. *Computer Communications*, 36(6):656–665.
- 1069 Shenker, S., Clark, D., Estrin, D., and Herzog, S. (1996). Pricing in computer networks: Reshaping the
1070 research agenda. *ACM SIGCOMM Computer Communication Review*, 26(2):19–43.
- 1071 Sidki, L., Ben-Shimol, Y., and Sadoski, A. (2016). Fault tolerant mechanisms for sdn controllers. In
1072 *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*,

- 1073 pages 173–178. IEEE.
- 1074 Sinha, Y. and Haribabu, K. (2017). A survey: Hybrid sdn. *Journal of Network and Computer Applications*,
1075 100:35–55.
- 1076 Smith, J. M., Farber, D. J., Gunter, C. A., Nettles, S. M., Feldmeier, D., and Sincoskie, W. D. (1996).
1077 Switchware: accelerating network evolution (white paper).
- 1078 Staessens, D., Sharma, S., Colle, D., Pickavet, M., and Demeester, P. (2011). Software defined networking:
1079 Meeting carrier grade requirements. In *Local & Metropolitan Area Networks (LANMAN), 2011 18th*
1080 *IEEE Workshop on*, pages 1–6. IEEE.
- 1081 Stephens, B., Cox, A. L., and Rixner, S. (2013). Plinko: Building provably resilient forwarding tables. In
1082 *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, page 26. ACM.
- 1083 Stephens, B., Cox, A. L., and Rixner, S. (2016). Scalable multi-failure fast failover via forwarding table
1084 compression. In *Proceedings of the Symposium on SDN Research*, page 9. ACM.
- 1085 Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., and Minden, G. J. (1997). A survey
1086 of active network research. *IEEE communications Magazine*, 35(1):80–86.
- 1087 Tennenhouse, D. L. and Wetherall, D. J. (2002). Towards an active network architecture. In *DARPA*
1088 *Active Networks Conference and Exposition, 2002. Proceedings*, pages 2–15. IEEE.
- 1089 Thorat, P., Challa, R., Raza, S. M., Kim, D. S., and Choo, H. (2016). Proactive failure recovery scheme
1090 for data traffic in software defined networks. In *NetSoft Conference and Workshops (NetSoft), 2016*
1091 *IEEE*, pages 219–225. IEEE.
- 1092 Thorat, P., Raza, S., Kim, D. S., and Choo, H. (2017). Rapid recovery from link failures in software-defined
1093 networks. *Journal of Communications and Networks*, 19(6):648–665.
- 1094 Thorat, P., Raza, S. M., Nguyen, D. T., Im, G., Choo, H., and Kim, D. S. (2015). Optimized self-healing
1095 framework for software defined networks. In *Proceedings of the 9th International Conference on*
1096 *Ubiquitous Information Management and Communication*, page 7. ACM.
- 1097 Tilmans, O. and Vissicchio, S. (2014). Igp-as-a-backup for robust sdn networks. In *Network and Service*
1098 *Management (CNSM), 2014 10th International Conference on*, pages 127–135. IEEE.
- 1099 Tran-Gia, P. (2018). Estimating the flow rule installation time of sdn switches when facing control plane
1100 delay. In *Measurement, Modelling and Evaluation of Computing Systems: 19th International GI/ITG*
1101 *Conference, MMB 2018, Erlangen, Germany, February 26-28, 2018, Proceedings*, volume 10740, page
1102 113. Springer.
- 1103 Trema (2011). Full-stack openflow framework in ruby and c.
- 1104 Van Adrichem, N. L., Iqbal, F., and Kuipers, F. A. (2016). Backup rules in software-defined networks. In
1105 *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*,
1106 pages 179–185. IEEE.
- 1107 Van Adrichem, N. L., Van Asten, B. J., and Kuipers, F. A. (2014). Fast recovery in software-defined
1108 networks. In *Software Defined Networks (EWSN), 2014 Third European Workshop on*, pages 61–66.
1109 IEEE.
- 1110 Vasseur, J.-P., Pickavet, M., and Demeester, P. (2004). *Network recovery: Protection and Restoration of*
1111 *Optical, SONET-SDH, IP, and MPLS*. Elsevier.
- 1112 Vissicchio, S., Cittadini, L., Vissicchio, S., and Cittadini, L. (2017). Safe, efficient, and robust sdn
1113 updates by combining rule replacements and additions. *IEEE/ACM Transactions on Networking (TON)*,
1114 25(5):3102–3115.
- 1115 Wang, L., Yao, L., Xu, Z., Wu, G., and Obaidat, M. S. (2018). Cfr: A cooperative link failure recovery
1116 scheme in software-defined networks. *International Journal of Communication Systems*, 31(10):e3560.
- 1117 Wang, N., Ho, K., Pavlou, G., and Howarth, M. (2008). An overview of routing optimization for internet
1118 traffic engineering. *IEEE Communications Surveys & Tutorials*, 10(1).
- 1119 Watanabe, T., Omizo, T., Akiyama, T., and Iida, K. (2015). Resilientflow: deployments of distributed
1120 control channel maintenance modules to recover sdn from unexpected failures. In *Design of Reliable*
1121 *Communication Networks (DRCN), 2015 11th International Conference on the*, pages 211–218. IEEE.
- 1122 Wetherall, D. J., Gutttag, J. V., and Tennenhouse, D. L. (1998). Ants: A toolkit for building and dynamically
1123 deploying network protocols. In *Open Architectures and Network Programming, 1998 IEEE*, pages
1124 117–129. IEEE.
- 1125 Yang, H., Cheng, L., Yuan, J., Zhang, J., Zhao, Y., and Lee, Y. (2015). Multipath protection for data
1126 center services in openflow-based software defined elastic optical networks. *Optical Fiber Technology*,
1127 23:108–115.

- 1128 Yu, Y., Xin, L., Shanzhi, C., and Yan, W. (2011). A framework of using openflow to handle transient
1129 link failure. In *Transportation, Mechanical, and Electrical Engineering (TMEE), 2011 International*
1130 *Conference on*, pages 2050–2053. IEEE.
- 1131 Zhang, X., Yu, S., Xu, Z., Li, Y., Cheng, Z., and Zhou, W. (2017a). Flow entry sharing in protection
1132 design for software defined networks. In *GLOBECOM 2017-2017 IEEE Global Communications*
1133 *Conference*, pages 1–7. IEEE.
- 1134 Zhang, Y., Cui, L., Wang, W., and Zhang, Y. (2017b). A survey on software defined networking with
1135 multiple controllers. *Journal of Network and Computer Applications*.