IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 20, 2019, accepted July 31, 2019, date of publication September 2, 2019, date of current version September 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2939115

Fault-Tolerance in the Scope of Software-Defined Networking (SDN)

A. U. REHMAN, RUI. L. AGUIAR, AND JOÃO PAULO BARRACA
Instituto de Telecomunicações, P-3810-193 Aveiro, Portugal
Departamento de Eletrónica, Telecomunicações e Informática, Universidade de Aveiro, Campus Universitário de Santiago, P-3810-193 Aveiro, Portugal

Corresponding author: A. U. Rehman (asad.rehman@av.it.pt)

This work is funded by Fundação para a Ciência e a Tecnologia/Ministério da Educação e Ciência (FCT/MEC) through national funds and when applicable co-funded by FEDER - PT2020 partnership agreement under the project UID/EEA/50008/2019, and Fundação para a Ciencia e Tecnologia under Grant PD/BD/113822/2015.

ABSTRACT Fault-tolerance is an essential aspect of network resilience. Fault-tolerance mechanisms are required to ensure high availability and high reliability in systems. The advent of software-defined networking (SDN) has both presented new challenges and opened new paths to develop novel strategies, architectures, and standards to support fault-tolerance. In this survey, we address SDN fault-tolerance and discuss the OpenFlow fault-tolerance support for failure recovery. We highlight the mechanism used for failure recovery in Carrier-grade networks that includes detection and recovery phases. Furthermore, we highlight SDN-specific fault-tolerance issues and provide a comprehensive overview of the state-of-the-art SDN fault-tolerance research efforts. We then discuss and structure SDN fault-tolerance research according to three distinct SDN planes (i.e., data, control, and application). Finally, we conclude enumerating future research directions for SDN fault-tolerance development.

INDEX TERMS Software-defined networking, fault-tolerance, OpenFlow, failure detection, failure recovery, fault-tolerance issues, network programmability, network softwarization, mission-critical communications.

I. INTRODUCTION

Due to the lack of software programmability in today's networks, it is quite challenging to modify (program) networks. Traditionally, there was no underlying programming abstraction provided to deal with the inherent complexity of distributed system failures. One of the primary features that software-defined networking (SDN) provides is data and control plane separation, laying the ground for simple network programmability. Although there is an extensive set of SDN research, most of the research performed so far focuses on exploring SDN as a programmatical technology, without considering fault-tolerance aspects [1]–[4].

Fault-tolerance is a broad area of knowledge, and covering all aspects of fault-tolerance concepts in a single paper is difficult. Hence, in this paper, we briefly discuss key fault-tolerance concepts and focus more on fault-tolerance in the scope of SDN. It is important to note that fault-tolerance and fault-management concepts are different. On the one hand, fault-tolerance is a characteristic of a system, which is designed in such a way that it can minimize service failures in the presence of system components faults. On the other hand "Fault management" is a term used in network management, describing the overall processes and infrastructure associated with detecting, diagnosing, and fixing faults, and returning to normal operations in telecommunication systems [5].

Generally, fault-tolerance is an essential part of the design of any communication system/network. Computer networks are built on physical infrastructure or virtualized versions of the physical infrastructure. These infrastructures are critical because business applications rely on their proper operation of such infrastructures. However, such infrastructures are prone to a wide range of challenges and attacks such as natural disasters or Denial of Service (DoS) attacks and major issues such as faults, failures, and errors all of which cause failure and disruption in network service. Therefore, to overcome these network service issues, resilience procedures and fault-tolerance mechanisms are essential to identify and heal the system/network in the presence of such failures [6].

SDN provides network flexibility through a clear separation of control and data planes, inherently simplifying network management [7], although SDN fault-tolerance is still in

The associate editor coordinating the review of this article and approving it for publication was Mubashir Husain Rehmani.

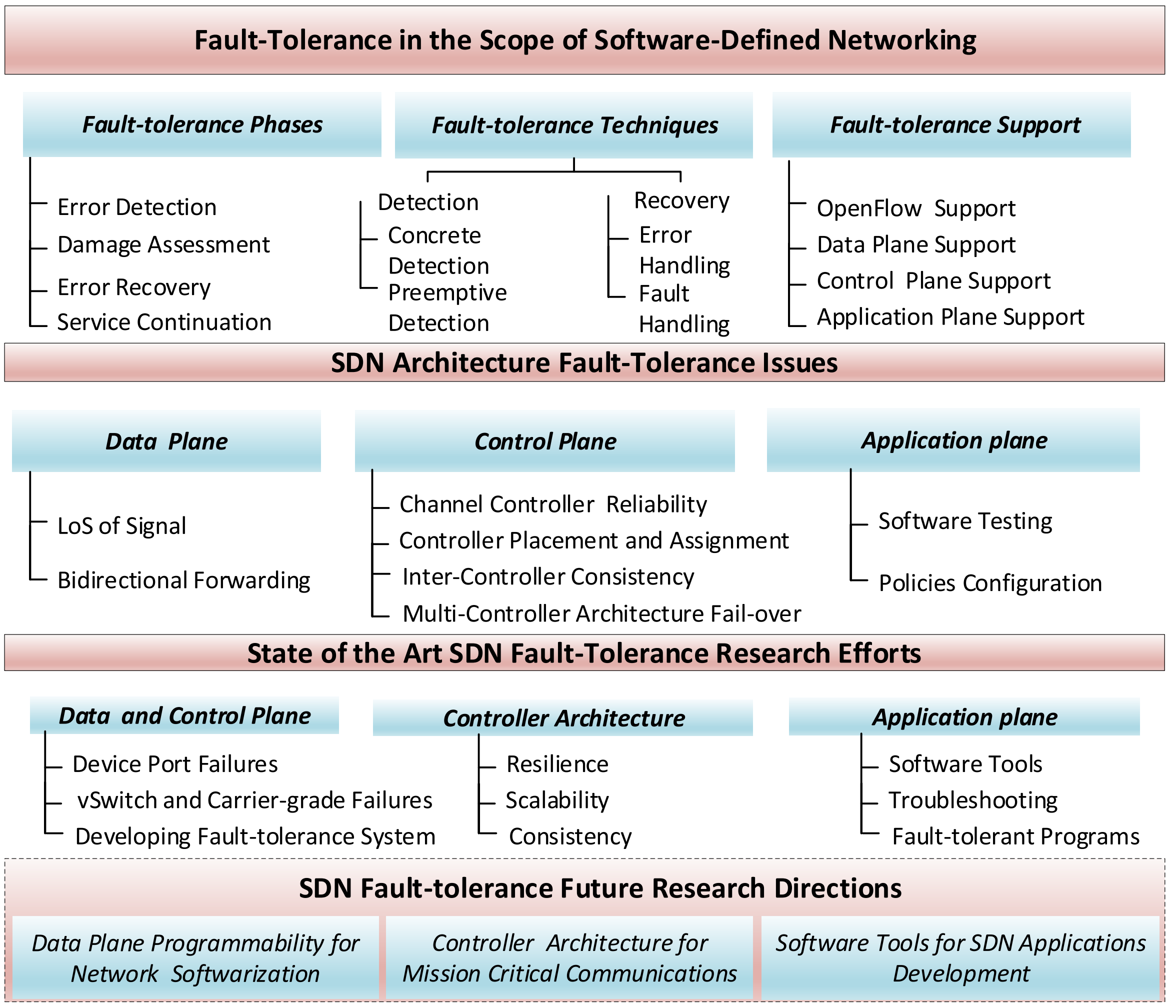
Fault-Tolerance in the Scope of Software-Defined Networking
Fault-tolerance Phases
Error Detection
Damage Assessment
Error Recovery
Service Continuation
Fault-tolerance Techniques
Detection
Concrete Detection
Preemptive Detection
Recovery
Error Handling
Fault Handling
Fault-tolerance Support
OpenFlow Support
Data Plane Support
Control Plane Support
Application Plane Support
SDN Architecture Fault-Tolerance Issues
Data Plane
LoS of Signal
Bidirectional Forwarding
Control Plane
Channel Controller Reliability
Controller Placement and Assignment
Inter-Controller Consistency
Multi-Controller Architecture Fail-over
Application plane
Software Testing
Policies Configuration
State of the Art SDN Fault-Tolerance Research Efforts
Data and Control Plane
Device Port Failures
vSwitch and Carrier-grade Failures
Developing Fault-tolerance System
Controller Architecture
Resilience
Scalability
Consistency
Application plane
Software Tools
Troubleshooting
Fault-tolerant Programs
SDN Fault-tolerance Future Research Directions
Data Plane Programmability for Network Softwarization
Controller Architecture for Mission Critical Communications
Software Tools for SDN Applications Development

FIGURE 1. **Condensed structure of this survey.**

its infancy. SDN is exposed to new sets of failures and issues at each layer of its architecture, as discussed in Section V. It is necessary to address these issues and safeguard each layer of the SDN architecture to provide enhanced fault-tolerance. We overview fault-tolerance, its techniques, and its typical phases of fault-tolerance. We then highlight fault-tolerance issues according to the SDN three main layers (data, control, and application) and classify SDN fault-tolerance research according to these three layers. The condensed structure of this survey is depicted in Fig. 1.

The rest of the paper is structured as follows. Section II discusses previous studies generic and specific to SDN fault-tolerance. Section III provides a comprehensive overview of fault-tolerance. Section IV summarizes the SDN layered architecture and fault-tolerance support in SDN as mentioned, Section V discusses briefly SDN architecture based fault-tolerance issues focusing on three principal planes of SDN architecture: data plane, control plane, and application plane. Section VI provides state-of-the-art of SDN fault-tolerance research efforts. Section VII discusses perspective SDN fault-tolerance challenges and future research directions. Section VIII concludes the paper. The list of abbreviations/acronyms is provided after the conclusion.

II. RELATED WORK

Some previous surveys have explored fault management [8], [9] in SDN. We provide further discussions on these efforts as follows:

Fonseca and Mota [8] addressed fault management in SDN. They presented an SDN fault management overview and focused more on issues associated with each layer of the SDN architecture. They also discussed general approaches, trade-offs, major contributions, and research gaps. They further extended the discussion on SDN fault management issues to optical and wireless networks.

Yu *et al.* [9] also addressed SDN fault management and provided a systematic survey by evaluating existing SDN fault management solutions. Furthermore, they presented an in-depth analysis of SDN fault management focusing system monitoring, fault diagnosis, fault recovery, and repair and

TABLE 1. Scope and technical contributions of previous and this work.

Related Work	Scope of the Work	Main Technical Contributions
Fault management in software-defined networks [8].	Provide a comprehensive review of fault management in software-defined networks and address fault-tolerance issues that can cause faults in each plane of SDN.	Identified main fault management issues in SDN, classified and discussed research efforts that addressed fault management in SDN. Furthermore, highlighted open issues and identified research gaps in SDN fault management research.
Fault management in software-defined networking [9].	Provide a systematic literature review by evaluating existing SDN fault management solutions as identified through advancements in both the research community and industry.	Presented an in-depth analysis of SDN fault management focusing on system monitoring, fault diagnosis, and fault recovery and repair. Compared and analyzed existing solutions in the context of SDN fault management over the period 2008–2017.
When software-defined networks meet fault-tolerance [31].	Study traditional fault-tolerance approaches and analyze their connections with SDN. Briefly discuss and compare data plane failure detection and recovery mechanisms (link/node).	Discussed traditional fault-tolerance approaches and compared restoration and protection methods for link/node failure recovery.
Fault-tolerance in the scope of software-defined networking (this work).	Previous studies do not discuss fault-tolerance phases, techniques or basic topics in the context of SDN architecture. This survey addresses fault tolerance specifically in the context of the SDN architecture.	Structured SDN fault tolerance according to the SDN layer-based architecture (i.e., data, control, and application planes). Provided a comprehensive overview of fault-tolerance issues on each SDN plane and discussed state-of-the-art research efforts addressing these. Highlighted SDN fault-tolerance issues and outlined important future research directions.

briefly discussed SDN fault-tolerance. They compared and analyzed existing solutions in the context of SDN fault management over the period of 2008-2017.

Chen *et al.* [31] addressed traditional fault-tolerance approaches and analyzed their connections with SDN. Data-plane failure detection and recovery mechanisms (link/node) were compared and briefly discussed. Furthermore, they discussed traditional fault-tolerance approaches and compared restoration and protection methods for link/node failure recovery.

In this paper, we focus on one of the utmost important disciplines of resilience, namely, fault-tolerance. We characterize it according to the SDN planes (data, control, and application). Furthermore, Table 2 locates the present work in the context of other technical contributions.

A. CONTRIBUTION AND SCOPE OF THIS SURVEY

As mentioned in Section II, previous studies have not discussed fault-tolerance phases, techniques, and basic topics in the context of SDN architecture. This survey addresses fault-tolerance specific to SDN. We can distinguish our contribution in this paper in comparison to the other related work as follows:

- We focus on SDN fault-tolerance rather than fault-management of SDN.
- We structure SDN fault-tolerance according to the SDN layer-based architecture (i.e, data, control, and application planes).
- We discuss in detail SDN fault-tolerance support in the context of traditional approaches that can be applied to address fault-tolerance in the data, control, and application planes.
- We provide and organize a comprehensive overview of fault-tolerance issues on each SDN plane and discuss state-of-the-art research efforts addressing these highlighted SDN fault-tolerance issues.
- We highlight the realization of SDN fault-tolerance challenges according to the data, control, and application planes and outline important future research directions from the perspective of programmability and network softwarization.

In summary, in this paper, we present a fault-tolerance overview, as well as techniques and phases in the scope of the SDN and traditional fault-tolerance support for SDN. We then highlight issues that can affect fault-tolerance in each layer of the SDN architecture itself, after which we structure and organize state-of-the-art research efforts addressing these SDN fault-tolerance issues and opt for solutions to safeguard fault-tolerance in each layer of the SDN architecture. We also outline important future research directions for each SDN layer, i.e., the programmable data plane for network softwarization (data plane), controller architecture for mission-critical scenarios (control plane) and software tools for fault-tolerant SDN application development (application plane).

III. BACKGROUND AND RELATED CONCEPTS

In this section, we discuss fault-tolerance and its techniques and provide a brief overview of fault-tolerance in traditional networks and its relationship with dependability.

A. FAULT-TOLERANCE OVERVIEW

Any system is prone to some sorts of threats that affect the operation of the system. Moreover, in computer networking, both distributed and centralized network systems are also prone to three major issues: Failures, Errors, and Faults.

A failure happens if a system is unable to implement the specified function appropriately. This means the service deviates from its specifications. An error is caused because one or more of the sequences of system states deviate from the specified sequence, and can cause service disruption. Faults can cause errors and lead to single or multiple failures [10].

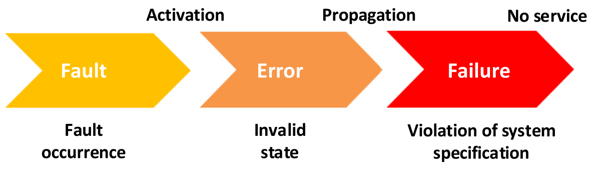

FIGURE 2. Relationship: Fault, Error, and Failure.

A fault is the hypothesized cause of an error, for instance, a software bug, human-made error or hardware power failure [11]. Relationship between fault, error, and failure is depicted in Fig. 2 [12].

Fault-tolerance is the outcome of a design process of building a reliable system from unreliable components [13]. Faults can be classified into two main categories [14], [15]: Crash faults and Byzantine faults. Crash faults can cause system fatal errors (for instance process and machine power-related failures), while Byzantine faults can cause the system to deviates from normal operation [14]. Fault-tolerance systems are equipped with several mechanisms that not only respond to these issues but also continuously offer and maintain correct system operation. However, it is hard to design a fault-tolerance system that can guarantee flawless communication in practice but, even in worst-case scenarios, fault-tolerance systems still offer graceful degradation of services. Nevertheless, we can always design efficient mechanisms for faults and errors that are most likely to happen and affect any system. Such approach can improve and enhance fault-tolerance communication systems.

B. FAULT-TOLERANCE PHASES

The typical four main stages of a fault-tolerance are as follows [16]:

1) **Error Detection:** In this stage, faults are first detected and then reported to determine the root cause of failure (observing failures).
2) **Damage Confinement and Assessment:** In this stage, the damaged or corrupted state of the system is assessed to determine the extent of the damage caused by faulty components.
3) **Error Recovery:** In this stage, recovery strategies are imposed to restore the system to a consistent and fault-free state. There are two different kinds of recovery techniques used:
   - Backward Recovery: In this technique, system states are recorded and stored so that a corrupted state can be discarded and the system can be restored to the previous fault free (correct) state.
   - Forward Recovery: In this technique, the system is being brought to a new correct fault-free state from the current corrupted state.
4) **Fault Treatment and Service Continuation:** In this stage, the location of faults are identified first and then faults are either repaired or the system is reconfigured to avoid faults. Service continuation is essential to ensure that the system will perform its operation normally and without immediate manifestation of faults.

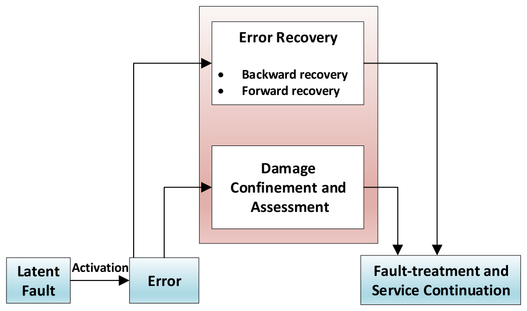

FIGURE 3. Typical phases in Fault-tolerance.

C. FAULT-TOLERANCE TECHNIQUES

Several fault-tolerance techniques are being used to avoid service failure in the presence of faults [17]. Fault-tolerance is carried out through error detection and system recovery, or simply detection and recovery mechanisms. Error detection identifies the presence of an error, while "recovery transforms a system state that contains one or more errors and (possibly) faults into a state without detected errors and faults that can be activated again " [18]. Recovery techniques can be further classified into two main categories: i) recovery with error handling; which eliminates errors from the system state; and ii) recovery with fault-handling; which prevents faults from being activated again. The choice of error detection and recovery techniques are being adopted based upon the underlying fault assumption. In the context of SDN, these fault-tolerance techniques must be explored in order to enhance fault-tolerance in future SDN environments.

The taxonomy of fault-tolerance techniques can be seen in Fig. 4, Table 2 [11] summarizes the details of such fault-tolerance techniques.

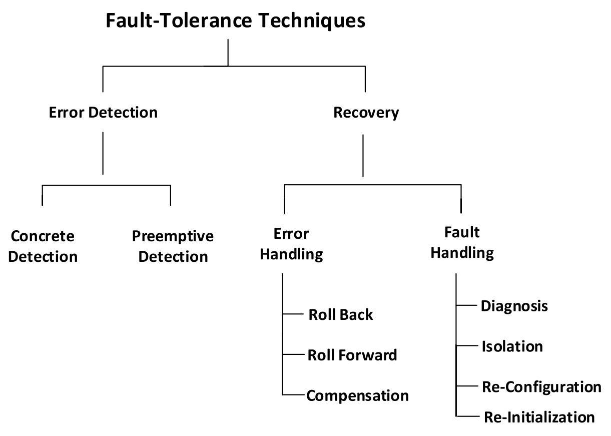

FIGURE 4. Fault-tolerance techniques.

TABLE 2. Details of Fault-tolerance techniques.

Error Detection	Recovery (Error Handling)	Recovery (Fault Handling)
Concurrent Detection: Occur during normal service delivery	Rollback: Restores system state to a saved last good known configurations before error occurrence	Diagnosis: Identify both error localization and its types
Preemptive Detection: Occur while normal delivery service is suspended; to check latent errors and dormant faults	Rollforward: Initiate a new system state without detected errors	Isolation: prevent the participation of faulty components that can leads to service failure
-	Compensation: Recover system erroneous state by enabling error to be masked through redundancy	Reconfiguration: Reassigns tasks among non-failed components
-	-	Reinitialization: Check and update system based on new configurations

IV. FAULT-TOLERANCE IN SDN

In this section, we provide an overview of SDN architecture and discuss SDN fault-tolerance based on the SDN architecture as divided into three main layers: data plane, control plane, and application plane.

A. SDN ARCHITECTURE OVERVIEW

SDN is a hot research topic, but there is increasing confusion regarding SDN concepts; architecture, multiple SDN networking planes, and interaction between layers through interfaces. Briefly, we discuss the SDN architecture and discuss the abstracted view of SDN planes.

The SDN architecture is shown in Fig. 5, which comprises several abstraction layers (abstraction of well-defined planes), interfaces (standardized Application Programmable Interfaces (APIs) between planes) and well-defined planes (collection of functions and resources with the same functionality) [19].

The three distinct SDN planes are as following:

1) **Data Plane:** The data plane (also known as forwarding plane) is responsible for handling data packets sent by the end-user through network devices that are responsible for traffic forwarding (based on instructions received from control plane).
The Forwarding Information Base (FIB), also known as forwarding table and Medium Access Control (MAC) table for routers and switches. FIB is used in the data plane to perform IP forwarding of unlabeled packets [20].
2) **Control Plane:** The control plane is responsible for deciding on how packets must be handled and forwarded at network devices to properly cross the network. The primary purpose of the control plane is to synchronize and update forwarding tables, while packet handling policies reside in the forwarding plane.
3) **Application Plane:** The plane where applications and services that define network behavior reside. Applications that directly (or primarily) support the operation of the forwarding plane (such as routing processes within the control plane) are not considered part of the application plane.

B. CONTROLLER

In SDN architectures, the controller is a logically centralized entity. It is responsible for translating the SDN applications requirement, via a Northbound API, down to the SDN data layer. Furthermore, it is also responsible for providing SDN applications an abstracted view of the network (including statistics and events).

Networks in SDN are managed by an external controller to process the flow of packets. This enables the programming of the network to be centrally controlled. Hence, the entire network and its devices can be managed efficiently regardless of the complexity of the underlying infrastructure. Moreover, SDN offers the flexibility through programming to separate the data and control planes with the logically centralized controller and by this, it is possible to modify the packet forwarding as per the network needs [1]. The OpenFlow standard has been proposed to manage the communication flow between the controller and network entities [21].

It is important to understand that different multi-controller architectures can exist with SDN. Bilal *et al.* [22] describe different types of SDN architecture and existing implementations. They further classify multi-controller SDN architecture in two broad categories (logically centralized and logically distributed architecture) and discuss in details with an example of implementation of such designs.

SDN controller fault-tolerance issues still exist and are addressed in current research, but are still far from providing the optimal solutions. Later in Section VI, we structure the SDN controller (centralized and distributed architecture) fault-tolerance research efforts.

C. SOUTHBOUND AND NORTHBOUND APIS

All the SDN networking planes are connected through specific interfaces, that standardized and simplify intercommunication between them. Intercommunication between SDN networking planes can be achieved in two different ways depending on the SDN architecture design, and the location of network entities. On one hand, if they are placed in a different location, a network protocol is used to provide communication interaction between them. On the other hand, if the network entities reside inside the same physical or virtual location, a communication interaction between network

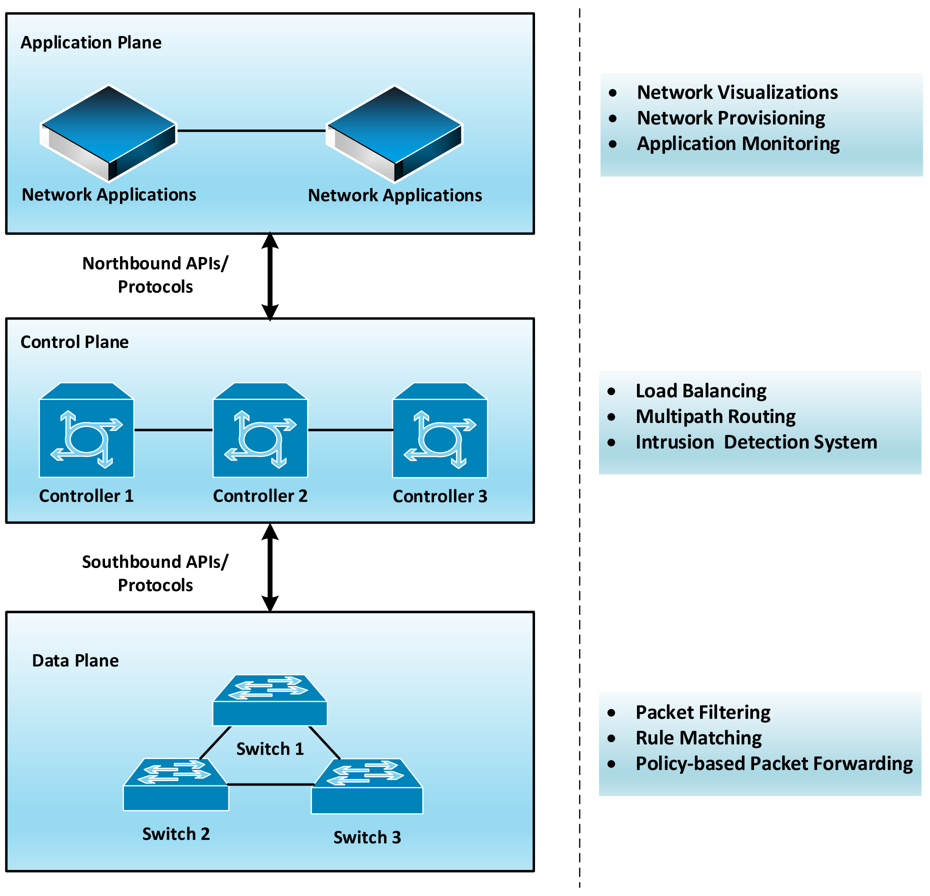

FIGURE 5. SDN high-level architecture: SDN planes and communication interfaces.

entities is possible with APIs. This enables the flexibility to design and implement intercommunication between network entities either through network protocols and/or APIs.

The SDN architecture has two primary interfaces (which use either APIs and/or protocols), as depicted in Fig. 5 [23], to enable intercommunication between two different SDN Planes: The Southbound and Northbound. In SDN terminology they are often referred to as Southbound APIs and Northbound APIs.

The Southbound API is a communication interface between the data and control plane. Currently, OpenFlow is a default standard for this communication. Furthermore, in SDN, OpenFlow is not the only available protocol for Southbound interface [24]. Other protocols and/or APIs for Southbound interface are: Forwarding and Control Element Separation (ForCES) [25], Network Configuration Protocol (NETCONF) [26], and Extensible Messaging and Presence Protocol (XMPP) [27], but they are more rarely used.

The Northbound API is a communication interface between the control plane and the application plane. Currently, there is no standardized northbound API. Because of this, the development of network applications for SDN has not been accelerated [28]. Nevertheless, most implementations use REpresentational State Transfer (REST)-based API because it is platform and language independent [29].

D. OPENFLOW FAULT-TOLERANCE SUPPORT IN SDN

This section discusses OpenFlow fault-tolerance from the point of view of the requirement for Carrier-grade networks. Carrier-grade networks usually provide faster recovery against service failure (i.e recover within 50ms) [30]. If service is not able to recover within this time, then service providers may be jeopardizing their business. OpenFlow was developed to support communication with non-proprietary FIBs. The OpenFlow protocol provides an abstraction of FIB through the OpenFlow group table concept. Moreover, the OpenFlow protocol communicates with the controller, which can trigger modifications in packets forwarding rules. This makes the FIB programmable through OpenFlow.

In SDN networks, operations rely on the proper functioning of the controller. The control plane in SDN manages the control logic of switches. The control logic is critical in SDN based networks. This problem is minimized in the latest version of the OpenFlow protocol by a master-slave configuration at the control layer: to increase resiliency. However, we argue that a tight synchronization must be required between a master and slave configuration to maintain an identical state of this configuration or the same copy of the master controller and this causes extra overhead in networks and sophisticated network management demands. It is quite challenging, to reach the recovery time equivalent to the standards set by the Carrier-grade networks, therefore, in order to enhance OpenFlow fault-tolerance support, mechanisms that not only maintain controller persistent state but also provides efficient recovery in case of controller fail-over must be developed. Another research challenge is the optimization of recovery time as per the Carrier-grade requirement as well as scalability. Indeed, Carrier-grade networks are a network of networks and scalability is a critical aspect. This excludes any solution not scalable, as such solution is not of interest for such Carriers.

E. SDN DATA PLANE FAULT-TOLERANCE SUPPORT

SDN data plane fault-tolerance is related to the issues already present in traditional architectures (e.g. Multiprotocol Label Switching technology). Due to the static nature of traditional networks, these approaches can achieve good performance upon link and node failures. However, failure detection and recovery approaches in dynamic networks such as SDN must be re-designed to adapt to the dynamics of the rapidly changing networks. Traditionally, reactive and proactive approaches were used to provide Fault-tolerance [31]. In the reactive approach, an alternative path is calculated after the fault becomes active. In proactive techniques, the resources and backup paths are pre-programmed before the occurrence of a fault (when a fault is dormant). If the fault becomes active, the pre-programmed logic starts to defend immediately and recover the system from faults. In this section, we address such failure detection and recovery approaches.

1) FAILURE DETECTION APPROACHES

The high availability of the data plane plays an important role to maintain the required communication from source to destination. To achieve high resiliency in the data plane, two steps are required: first, design and analyze the topology in the presence of known and unknown failures; and, second, to design an alternative path according to the type of failures that occur in the network. In Carrier-grade networks, two well-known mechanisms exist to detect failures in the data plane, namely Loss of Signal (LOS) and Bidirectional Forwarding Detection (BFD) [32]. LOS detect failures in a specific port of the forwarding device, while BFD can detect path failure between any two forwarding devices. Both methods provide failure detection at an accelerated rate, independent of the media type and routing protocols (such as Open Shortest Path First (OSPF) and Enhanced Interior Gateway Routing Protocol (EIGRP)).

2) FAILURE RECOVERY APPROACHES

In Carrier-grade networks the recovery mechanism must guarantee the recovery process within 50 ms [33]. For this purpose, restoration and protection are widely used to recover from network service failures — methods based on reactive and proactive approaches. Protection is classified as a reactive technology while restoration is classified as pro-active technology. In restoration, an alternative path is only established after the occurrence of failure and resources are not reserved before the occurrence of the failure, and the paths are pre-assigned or allocated dynamically. However, in the case of protection, the alternative paths are already reserved and assigned before the occurrence of a failure. This needed no added processing (signaling) to recover from failure. In restoration, additional signaling is needed to recover from failure; in large networks, this is not often possible within the set requirement of Carrier-grade networks, and thus it is not scalable. However, in protection, as a matter of fact, the additional signaling is not required, and recovery process is fast when compared to restoration, with the recovery process possible within 50 ms and suitable for Carrier-grade networks.

F. SDN CONTROL PLANE FAULT-TOLERANCE SUPPORT

Control plane resilience is a requirement for proper operation in networks: the controller is vital, which means that the controller must be able to process all required traffic commands in all situations. There are several approaches to enhance SDN control plane fault-tolerance. The first approach is to replicate a controller on a different control network. In the case of failure, the replicated controller takes over and manages traffic. In another approach, the controller must be embedded with mechanisms (build-in module) to self-heal from targeted attacks such as Denial of Service (DoS), flooding and fake traffic routing and other network- related targeted attacks. However, the control plane time to recover from such attacks is critical, and ideally, recovery mechanisms must be developed to mitigate failures within the set network requirements. In addition to these, the recovery process must be efficient and must be able to self-heal during a failure event with minimum overhead. In-band and out-of-band signaling solutions have been adopted to offer SDN control plane reliability [34]. In practice, most SDN deployments use out-of-band control, where control packets are managed by a dedicated management network [35].

G. SDN APPLICATION PLANE FAULT-TOLERANCE SUPPORT

On an SDN network, the Application plane is the layer that has applications and services that make requests for network functions provided by the control plane and the data plane. On traditional networks, security, management, and monitoring devices or applications reside in this layer.

The application layer allows business applications to modify and influence the way the network behaves in order to provide services to customers. This requires the definition of an API, to allow third-party developers to build and sell network applications to the network operator. The development of such an API has not yet properly addressed by the Open Network Foundation (ONF) but is required in order to guarantee interoperability between a business application and network controllers from different suppliers.

Existing SDN programming languages offer several features such as flow installation, policy definition, programming paradigm and abstraction for developing and enabling network and application fault-tolerance in SDN [36]–[39].

V. SDN ARCHITECTURE FAULT-TOLERANCE ISSUES

In this section, first, we highlighted SDN fault-tolerance issues and then provide state-of-the-art research efforts focusing on such fault-tolerance issues in SDN. Furthermore, we structure them based on three main layers of SDN architecture. These are later summarized in Table3, Table 4, and Table 5.

A. DATA PLANE ISSUES

There are two main data plane layer issues namely: network failure detection and recovery. These issues arise either due to link or node failure. As discussed, in traditional networks, to detect network failure, a particular protocol, such as LOS and BFD, is used [40]. Also, to recover from network failure, restoration and protection approaches are widely used. However, resolving these issues in the SDN environment is challenging due to the centralized nature of the controller. For instance, in the SDN-based environment, the controller can take a longer time to detect and recover from link or node failure due to the rapid changing abstracted view of the network (dynamic topology). Therefore, there is a need to develop mechanisms for SDN that can provide faster recovery [40].

B. CONTROL PLANE ISSUES

There are multiple SDN control plane issues. The three main issues that are critical to SDN control plane fault-tolerance can be classified as:

1) CONTROLLER CHANNEL RELIABILITY

In SDN controllers, communications with underlying devices are critical. Therefore, their availability is a must condition to protect the proper operations of a network. The controller channel must be fault-tolerant (reliable) in-case of failure due to loss of switches connection, or error due to the communications protocols between the controller and underlying devices. These issues can disrupt the network and lead to several failures in the SDN network. In order to cope with these issues, controller redundancy [41], [42] and path backup are considered essential.

2) CONTROLLER PLACEMENT AND ASSIGNMENT

Controller placement (how to choose the location of controllers) and assignment (how to assign the controllers to the switches) are two significant issues [43], generally known as the controller placement problem [44].

The controller's assignment issue (balance of controllers) in SDN is important, not only from the point of view of fault-tolerant controller design but also from the point of view of network optimization. Improper controller assignment can lead to two main problems: i) under-provisioning: When a small number of controllers are placed to handle more traffic than its capacity of processing. In this case, the controller is overloaded and possibly increases downtime and affects network performance, and 2) Over-provisioning: When more than the required controllers are placed to handle comparably low traffic environment. In this case, costly controllers are underutilized.

To deal with the controller placement issue, one of the strategies is to develop algorithms that can provide optimal controller placement in dynamic SDN-based networks, which is also challenging in itself [45].

3) INTER-CONTROLLER CONSISTENCY

In order to avoid a single point of failure in SDN networks, multi-controllers architecture approaches are pursued, (either in physically centralized and logically distributed, or fully distributed fashion with the coordination of different SDN controllers) [46]. It is important to note that these practices increase resiliency, but there is a strict requirement for controllers consistency [47]. The level of consistency depends on stateful or stateless backup settings. The controller must maintain a persistence state to guarantee controller consistency.

4) MULTI-CONTROLLER ARCHITECTURE FAIL-OVER

In SDN, multi-controller architectures can follow the flat/horizontal or hierarchical/vertical designs. On the one hand, in a flat architecture, the control plane has just one layer, and each controller has the same responsibilities [22]. The advantage of such architecture design is that it provides more resilience against failure, but the task of managing controllers is difficult. On the other hand, in a hierarchical architecture, the control plane has multiple layers, and each controller has different responsibilities (due to multiple level partitioning). The advantage of such design is that it provides a more straightforward way to manage controllers. Both of these multi-controller architecture approaches can be used to improve switch to controller latency or vice versa. In both designs, it is important to consider that controllers must respond to any fail-over [48] request efficiently and without affecting the performance.

C. APPLICATION PLANE ISSUES

SDN enables programmability to control network devices more efficiently but this is highly dependent on the quality of software development. In order to develop reliable SDN applications, debugging (the process followed to fix bugs) and testing (verification) tools can not only advance software quality but also help in fixing software bugs as service evolves (continuous development process) [49]. To ensure the quality of software network troubleshooting, debugging and testing are consider essential [37].

Network visualization, network provisioning, and application monitoring can be conceptualized as an SDN application layer. For this reason, fault-tolerance of both network and applications can be supported at the application plane. Moreover, in order to develop fault-tolerant network applications, all the phases from application design to final application deployments must undergo proper testing. Currently, there are certain languages proposed that enable the construction of fault-tolerant programs to write SDN-based fault-tolerant systems. Since fault-tolerance of both network and applications can be supported at application plane, the two main SDN application layer issues are as follows:

1) SOFTWARE TESTING

The network behavior in a SDN-based network is controlled by a set of software programs. For proper network troubleshooting support, the SDN applications must be resilient [50]. Resilient design help to the identify the root cause of the bug and the administrator can then track and isolate faults so that the system can be restored to the correct operating state.

2) POLICIES CONFIGURATION

In SDN, network management becomes more dependent on software development due to programmability. There is a risk that policies across the network can be violated due to untested errors (bugs) in the application, which can be propagated to affect SDN controllers, protocols and routing policies and eventually can lead to network service failure. Therefore, constant application monitoring is essential to avoid any violation of network policies [51].

VI. SDN FAULT-TOLERANCE RESEARCH EFFORTS

SDN offers greater flexibility and network automation when compared with traditional distributed systems, at the risk of the controller being a single point of failure.

Most of the research carried out has been focused on exploring these technologies rather than evaluating the associated reliability aspects. In recent years, a shift is being made towards the evaluation of SDN fault-tolerance. We reviewed the research studies carried out that address SDN data, control, and application planes fault-tolerance. The details are summarized in Table3, Table 4, and Table 5. Furthermore, classification of SDN state-of-the-art research efforts according to SDN principal planes and controller architecture is depicted in Fig. 6.

A. DATA AND CONTROL PLANES

In this section, we discuss main research efforts that have addressed data and control plane fault-tolerance in the context of SDN.

Current fault-tolerance techniques are not yet proven to meet Carrier-grade fault-tolerance requirement (50 ms recovery time) [30]. A research study carried out by Sharma *et al.* [33] provided experimental evidence that protection provides faster recovery as compared to restoration and is thus more suitable to guarantee resilience in large scalable networks [52], [53].

Adrichem *et al.* [40] argued that time was a critical metric in the recovery process during network failures. It is still difficult to develop mechanisms that guarantee efficient recovery. In their research study, they demonstrated that current failure recovery approaches (restoration and protection) suffered from long delays. They introduced a failover scheme based on a per-link BFD approach and showed that implementation reduced recovery time. They performed experiments to evaluate different network topologies and showed that recovery time was consistent irrespective of network size.

Mohan *et al.* [54] carried-out a research study to provide fault-tolerance in the specific case of Ternary Content Addressable Memory (TCAM) limited SDN. They argue that proactive fault-tolerance policies provide faster failure recovery based on restoring the re-routing paths. This requires large forwarding rules to be installed on the TCAM, but TCAM has limited memory. Based on these challenges, they have developed an optimized programming formulation that determines the set of backup paths to protect a flow and minimize the number of forwarding rules for the backup paths to be installed in the switch TCAM. This means that fewer rules would be required for backup paths. They proposed two algorithms Backward Local Rerouting (BLR) and Forward Local Rerouting (FLR) [54] to improves TCAM and bandwidth usage efficiency for single link failure in SDN system.

Li *et al.* [55] carried research studies to enhanced failure recovery in SDN with customized control. They have developed a Declarative Failure Recovery System (DFRS) based on three algorithms backup path construction, add and subtract. The Backup paths construction algorithm creates safe backup paths based on the recovery demands. Further, it adds and subtracts algorithms to find a minimum number of the paths to be allocated to guarantee network services during failure with minimum memory overhead [55]. Three different topologies were evaluated to test the effectiveness and scalability of DFRS. They achieved similar performance to the traditional failure protection algorithm, but with 5% less backup rules. In the event of failure many switches are allocated hundreds of forwarding rule for backup; this burden the switch, affects the performance and delays failure recovery. The authors argue that the DFRS system only allocates dozens of forwarding rules to switches, as compared to the usual hundreds of forwarding rules. Thus, this leads to effective memory utilization and improved stability.

Kuźniar *et al.* [56] proposed Automatic Failure Recovery (AFRO) for SDN, an automated runtime system that recovers system failure in OpenFlow system. They argue that they extend the basic functionality of the controller program with additional controller agnostic modules that provide efficient recovery.

Kim *et al.* [57] proposed the SDN fault-tolerant system CORONET, and they argue that their proposed system provides recovery against multipath failure in a data plane. However, since the initial published work in 2012, no significant contribution was made after, although many evolutions represented on SDN architectures and protocols.

Schiff *et al.* [58] presented a model to design self-stabilizing distributed control planes for SDN and argue that their proposed technique provides a mechanism to deal with key challenges of a distributed system, such as bootstrapping and in-band control. Further, they implemented a plug and play SDN distributed control plane to support automatic topology discovery and management in dynamic networks. However, it is important to note that the self-stabilizing

TABLE 3. Selected work on SDN data and control plane Fault-tolerance efforts.

Fault-Tolerance Efforts by	SDN Plane(s)	Main Purpose
Sharma et al. [30]	Data plane	Check device port failures and consistently monitors failure detection between two forwarding links to support standardized failure detection methods such as BFD and LOS.
Adrichem et al. [40]	Data plane	Improve the speed of failure detection in Open vSwitch based implementation.
Sharma et al. [52], Sharma et al. [53]	Control plane	Meet failure recovery requirements for Carrier-grade networks.
Mohan et al. [54]	Control plane	Improve TCAM and bandwidth efficiency for single link failure in SDN system by reducing number of flow rules.
Li et al. [55]	Control plane	Compute different backup according to source and destination pairs.
Kuzniar et al. [56]	Control plane	Recover system failures in OpenFlow based controller implementations.
Kim et al. [57]	Control plane	Develop a fault-tolerant system, able to recover from multiple link failures in the data plane.
Schiff et al. [58]	Control plane	Presented a model to design self-stabilizing distributed SDN control planes.
Chen et al. [59]	Data and Control plane	Enable faster recovery with low memory using VLAN tagging concept.
Jain et al. [60]	Data and Control plane	Evaluate outage and failure in a private SDN WAN.

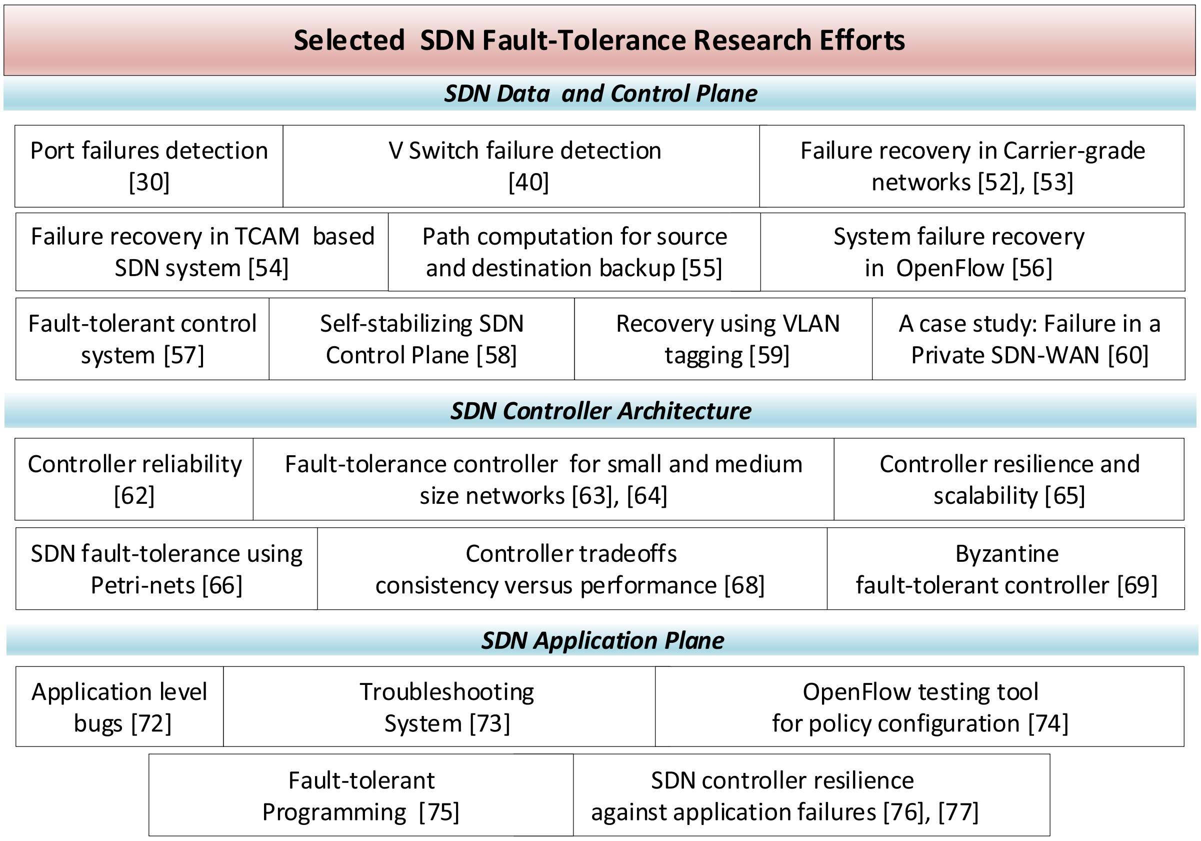
Selected SDN Fault-Tolerance Research Efforts
SDN Data and Control Plane
Port failures detection [30]
V Switch failure detection [40]
Failure recovery in Carrier-grade networks [52], [53]
Failure recovery in TCAM based SDN system [54]
Path computation for source and destination backup [55]
System failure recovery in OpenFlow [56]
Fault-tolerant control system [57]
Self-stabilizing SDN Control Plane [58]
Recovery using VLAN tagging [59]
A case study: Failure in a Private SDN-WAN [60]
SDN Controller Architecture
Controller reliability [62]
Fault-tolerance controller for small and medium size networks [63], [64]
Controller resilience and scalability [65]
SDN fault-tolerance using Petri-nets [66]
Controller tradeoffs consistency versus performance [68]
Byzantine fault-tolerant controller [69]
SDN Application Plane
Application level bugs [72]
Troubleshooting System [73]
OpenFlow testing tool for policy configuration [74]
Fault-tolerant Programming [75]
SDN controller resilience against application failures [76], [77]

FIGURE 6. SDN state-of-the-art research efforts: Principal Planes and controller architecture.

distributed plane is still at a very early stage, and a lot more effort is needed to step forward to a proof of concept stage. The authors also affirm that a feasibility study is needed to further validate their proposed model of the self-stabilizing SDN control plane. Formal proofs are required for this plug and play distributed model to be shown effective in meeting fault-tolerance requirements for SDN and future networks.

Chen *et al.* [59] proposed a method of protection-based recovery in SDN using Virtual Local Area Network (VLAN) tags. They argue that their proposed method provides faster recovery with low memory usage and without the participation of the controller to switch to backup paths. In their system, protection takes 20 ms while recovery on average takes 50 ms to restore from failures. Similarly, Thorat et. al. [61] proposed a proactive policy to achieve fast failure recovery using VLAN tags and claims that 99 % reduction in flow storage is achieved as well as fast failure recovery as set in Carrier-grade networks.

Jain *et al.* [60] carried out a study to address network outage and failures. They evaluated three years of production experience with B4, their own SDN enabled Wide Area Network (WAN) that connects Google's data center. They implemented fault-tolerance policies such as customized forwarding and dynamic relocation of bandwidth and alternative link recovery using OpenFlow. Generally, control plane protection is achieved through resource replication and replicas were placed on different physical servers. They have analyzed in their study that SDN enabled WAN served more traffic than public WAN and offered cost-effective bandwidth and nearly 100 % link utilization, enabling high availability of resources. However, they admit that bottlenecks in the bridging protocol from the control plane to the data plane exists and needs to be optimized to improve performance further. Improving this will offer superior fault-tolerance in future SDN based networks.

B. CONTROLLER ARCHITECTURE

In this section, we discuss key research efforts that have addressed controller architecture fault-tolerance in the context of SDN.

Katta *et al.* [62] studied the fault-tolerance of the controller under crash failures. They argued that to offer a logically centralized controller, it is necessary to maintain a consistent controller state and ensure switch states consistently during controller failure. Therefore, they have proposed Ravana, an SDN based fault-tolerant protocol that provides an abstraction of the logically centralized controller. Ravana handles the entire event processing cycle and ensures total event ordering across the entire system. This enables Ravana to correctly handle switch state and replicas without the need of restoring to rollbacks. Moreover, it mitigates control messages during controller failures this help in extending the control channel interface. Ravana provides a reliable distributed control for SDN. However, it does not provide support for richer fault models such as Byzantine failures, and it is limited to multithreaded control applications, and the scalability is also one of the tests that are not evaluated in Ravana protocol.

Botelho *et al.* [63], [64] carried-out research studies and implemented a prototype that integrates a Floodlight based distributed controller architecture to BFT-SMaRT (Byzantine Fault-tolerant (BFT) and State Machine Replication (SMR)), a replicated state machine library. This enables the consistency between an SDN-controller and their redundant backups stored in a shared database. In their work, three SDN applications (learning switch, load balancer, and device manager), with slight modifications, were tested to analyze the workloads these applications were generating and measure the performance. The result of their study shows that the data store is capable of handling large workloads, but to maintain a strong consistency of data there was an increase in latency and this impacted performance. Thus, the solution seems not to be scalable, they argue that an acceptable level of fault-tolerance was easy to achieve. Moreover, the authors also proposed a practical fault-tolerant SDN controller design for small and medium networks. A shared database is replicated that save all network state. This database is created using a Replicated State Machine (RSM), and in their previous research studies, they argue that the database meets the performance requirements for small and medium networks. They incorporate a cache in the controller that avoids failure smoothly without any additional coordination service.

A master and slave controller configuration is implemented by Fonseca *et al.* [65] in which the solution to offer control plan resilience is provided by integrating a Control Plane Recovery (CPR) module into a standard OpenFlow controller build upon NOX OpenFlow controller. CPR is a two-phase process, consisting of replication and recovery, and offer resilience against several types of failure in an SDN enabled centrally controlled networks. Similarly, the research studies carried-out by Tootonchain and Ganjali [67] introduce HyperFlow to provide control plane resilience. HyperFlow is a distributed event-based control plane, which is physically distributed but logically centralized. This enables the scalability, as well as ensure the benefits of centralized network control. They argue that HyperFlow [70] offers a scalable solution for control plane resilience in SDN enabled network.

TABLE 4. Selected work on SDN Controller Fault-tolerance Efforts.

Fault-Tolerance Efforts by	Controller Architecture	Main Purpose
Katta et al. [62]	Centralized	Develop reliable distributed Control plane for SDN controller.
Botelho et al. [63], [64]	Centralized	Develop fault-tolerant controller for small and medium size networks.
Fonseca et al. [65]	Centralized	Enable Control plane resilience and scalability.
Aly et al. [66]	Centralized	Petri-net based mathematical framework to enhance SDN fault-tolerance.
Tootoonchian et al. [67]	Distributed	Enable Control plane resilience and scalability.
Gonzalez et al. [68]	Distributed	Evaluate trade-off between consistency and performance in a fault-tolerant SDN platform.
ElDefrawy and Kaczmarek [69]	Distributed	Develop SDN controller that can tolerates Byzantine faults.

Aly and Kotb [66] used SDN-centralized architecture in which a master controller is connected to a set of slave-controllers. Based on this set-up, they proposed a new Petri-net based mathematical framework for SDN fault-tolerance and named the model $\mathrm{FTPN}_{\mathrm{SDN}}$. They claim that, in order to avoid service disruption, Petri-net capability functions were used to identify the next back-up controller in the event of controller failure. They also showed that transition time needed to take over another controller was reduced by 10%. They evaluated the performance of their proposed model, comparing it with the HyperFlow reference model, and claim that they were able to reduce the 12% packet delay.

Clearly, a single controller point of failure limits scalability and we argue that the several recent research studies carried out do not yet provide a mechanism to achieve high-level performance or fault-tolerance at scale in SDN based networks.

To deal with the challenge of SDN controller consistency and performance, Gonzalez *et al.* [68] proposed a method to improve consistency and performance by using some of the approaches from the recent study carried out by Katta et al [62]. They design a mechanism to provide better consistency and performance in master-slave SDN configuration. They consider the performance metric for the SDN controller based on controller latency and throughput. Their proposed solution provides consistency and performance close to the offered by a single SDN controller. However, they emphasize that a very reliable communication channel is a must between the master controller and the data store.

ElDefrawy and Kaczmarek [69] proposed a fault-tolerance SDN controller design that tolerates Byzantine faults. However, their controller design has not yet achieved high-level performance for large-scale deployments. Further, they argued that their controller design is feasible for constructing resilient networks. In this research study, they have designed and prototyped a Byzantine-fault-tolerant distributed SDN controller to tolerate malicious faults both in control and in data plane as described in Kreutz *et al.* [2]. Further, they integrated the two existing SDN byzantine vulnerable controller with the BFT-SMaRt, a tool for creating byzantine fault-tolerant system [71].

C. APPLICATION PLANE

In this section, we discuss key research efforts that have addressed application plane fault-tolerance in the context of SDN.

SDN offers the flexibility of network programmability but this raised an issue of software-based troubleshooting and debugging which need to be addressed, as discussed.

Heller *et al.* [72] proposed a structured troubleshooting approach by exploiting the SDN layered architecture. They aim to develop a tool that would identify bugs by systematically tracking the root cause of detected failures. This would save time in diagnosing and enable the network administrator to directly fix the problems. However, they have not proposed any system or framework. In a similar way, Scott *et al.* [73] also studied SDN troubleshooting and proposed the SDN troubleshooting system (STS). This system aims to optimize the debugging time by filtering events not correlated to the source of failure. They have demonstrated the feasibility of their proposed system and have tested five SDN control open-source platforms: ONOS (Java) [78], POX (Python) [79], NOX (C++) [80], Pyretic (Python) [81], and Floodlight (Java) [82]. They were able to identify seven new bugs in real-time, and debugged them using their proposed STS system, and showed that STS enhances the time-consuming process for debugging in SDN. Likewise, Canini *et al.* [74] built NICE, a troubleshooting tool for SDN. The state-space of the entire SDN system is explored through model checking. This approach provides a systematic way to test unmodified controller programs. This tool automates the testing of OpenFlow application based on model checking and concocts execution efficiently.

Reitblatt *et al.* [75] proposed FatTire, a high-level declarative language for writing fault-tolerant network programs in SDN. This high-level language aims to provide policy-based network management where SDN programmers can construct specific policies (for instance, data security and customized forwarding). Earlier work of Lui *et al.* [83] emphasize that connectivity must be realized as data plane service. This work fits together with FatTire for implementing policy abstractions. Similarly, a study by Suchara *et al.* [84] based on integrating fault-tolerance and traffic engineering possibly be used with FatTire. Likewise, the Flow-based Management Language (FML) [85] specify policies using a declarative language to enforce policies within the enterprise. For instance, Access Control Lists (ACLs), Virtual Local Area Networks (VLANs) and policy-based routing. This differs from FatTire as it does not provide fault-tolerance policy. Similarly, Kazemian *et al.* [51] introduce NetPlumber, a real-time tool for policy checking based on Header Space Analysis

TABLE 5. Selected work on SDN application plane Fault-tolerance efforts.

Fault-Tolerance Efforts by	SDN Plane(s)	Main Purpose
Heller et al. [72]	Application plane	Develop a tool to identify bugs based on the root cause of actual bugs.
Scott et al. [73]	Application plane	Develop troubleshooting system and framework for SDN.
Canini et. al. [74]	Application plane	Develop a testing tool for OpenFlow based SDN to detects any violation of network correctness policies.
Reitblatt et al. [75]	Application plane	Develop high level language to write fault-tolerant programs to implement network policies.
Chandrasekaran and Benson [76], Chandrasekaranet et al. [77]	Application plane	Make SDN controller and network resilient to SDN application failures.

(HSA). The authors argue that they have applied this tool to Google's SDN and Stanford backbone and analyzed that 50-500 μs on average were required for a rule update against a single policy.

Chandrasekaran *et al.* [76], [77] claim that they have developed a fault-tolerant SDN controller framework called LegoSDN. The authors aim to achieve recovery of SDN application against both deterministic and non-deterministic service failures. Extending this work further, the authors develop a prototype which isolates SDN application from one and another, as well as from controller, by running each application securely in a sandbox. Thus, all failures are restricted to its virtual isolated space.

VII. SDN FAULT-TOLERANCE: CHALLENGES AND FUTURE RESEARCH DIRECTIONS

In this section, we outline future directions for SDN fault-tolerance development from the perspective of its role in future intelligent programmable networks. The term programmability refers to control the set of action, rules or to enforce a policy by software intelligently. The programmability empowers to utilize multi-vendor hardware/devices with enhanced flexibility. Moreover, it enables customized scripting through programming languages to facilitate network administrators to enforce policy-based configuration on network devices/functions through APIs. Therefore, programmability is a pre-requisite for enabling network automation (a practice in which software automatically configure and test network devices) in a communications network [86].

A programmable network is flexible and re-configurable because most of the protocol stacks are implemented in software. Therefore, network upgrades to replace or configure the network protocols is possible without the interruption of the network operations [87], [88].

The term Network softwarization refers to the ''networking industry transformation for designing, deploying, implementing and maintaining network devices/network elements through software programming'' [89].

A. DATA PLANE PROGRAMMABILITY FOR NETWORK SOFTWARIZATION

A few research studies included in Table 3 focused on SDN data plane fault-tolerance using traditional failure detection approaches (i.e., BFD and LOS) and recovery approaches (i.e., restoration and protection), that were able to ensure Carrier-grade reliability. However, due to the emergence of new data plane specifications such as Programming Protocol Independent Packet Processors (P4) [90], and Protocol-Oblivious Forwarding (POF) [91] new paths toward the development of novel strategies and standards to support fault-tolerance opened up. Data plane programmability in SDN is the next step towards supporting a fast-growing trend of network programmability [92] and network softwarization (softwarization of future networks and services) [93]. Traditionally, the network data plane was designed to be configurable but with fixed forwarding logic (packet processing with pre-defined logic). However, the SDN programmable data plane should provide the flexibility to modify forwarding logic (customized packet processing). Concerning the SDN data plane, there are still error detection and recovery issues, which require careful consideration. For instance, the current probe-based testing solution takes a long time to generate probe packets, [94], [95] making consistency between control plane policies and data plane forwarding behaviors difficult. Furthermore, additional new pipelines are required in the switch data path for collecting traffic statistics [96]; this process itself can cause errors.

Due to these challenges, the idea towards data plane programmability has attracted significant interest from both academia and industry [97]. Recent research studies addressed SDN data plane programmability. New data plane specification (eg., P4 and POF) has been evolved which extend the feature of SDN beyond OpenFlow specifications [98]. These new data plane specification can optimized fault-management in SDN, thus improve SDN architecture fault-tolerance and reliability aspects. we believe that data plane programmability is an important area for future SDN development.

B. CONTROLLER ARCHITECTURE FOR MISSION CRITICAL COMMUNICATIONS

Controller fault-tolerance research studies included in Table 4 were focused on designing fault-tolerant SDN controller in scenarios where parameters such as throughput, packet loss, latency, jitter, and redundancy are more flexible than mission-critical communications (industrial networks and intelligent systems) [99], [100], where these parameters have more stringent demands. Mission-critical applications are common in different sectors including military, hospital, automotive safety, and air-traffic control systems [101]. Unfortunately, the research and development of fault-tolerant SDN controller for mission-critical applications have been overlooked. Not even the SDN fault-tolerant controller research efforts (other than mission-critical communication) are still not yet fully developed. Scalability, performance, and data consistency in SDN multi-controller architectures is still an area of intense investigation [70], [102]. There is a need to develop fault-tolerant SDN control network for mission-critical applications, where designing SDN controller for mission-critical applications is of significant importance and quite challenging hence, we believe that this topic should be addressed comprehensively in future research.

C. SOFTWARE TOOLS FOR SDN APPLICATIONS DEVELOPMENT

SDN fault-tolerance research studies included in Table 5 were focused on developing software tools for troubleshooting, writing fault-tolerant programs, and detect any network policy violations in application plane. However, the developed fault-tolerant software tools are still having many shortcomings, for instance, incomplete repair mechanisms, and high overhead for recovery [9]. Due to the diversity of network protocols for SDN Southbound and Northbound APIs, and underway standardization of these diverse protocols, the new SDN applications development has not been accelerated. Hence, the developed software tools have not been comprehensively tested and developed to support the diversity of network protocols used in SDN networks. There is a need to develop improved software tools in order to enable application plane fault-tolerance in future SDN deployments.

VIII. CONCLUSION

This work presents a survey on fault-tolerance in the scope of SDN. Also, we provided a simple background on fault-tolerance and related concepts to develop a complete understanding of the topic. Our goal was to identify SDN fault-tolerance requirements specific to the SDN architecture and discuss approaches that can be used to improve fault-tolerance in SDN.

Current SDN research efforts were structured according to the three main layers of SDN architecture and categorized them according to data, control, and application planes.

While exploring the topic of fault-tolerance in SDN, we have identified that each layer has its faults and fault-tolerance issues. This means that in order to achieve fault-tolerance different aspects and features are needed to be targeted, and no single-focused technology will be able to provide the reliability expected in commercial networks.

Recent research studies show that SDN can play a pivotal role in shaping and managing future dynamic networking environments, such as cloud-native networks, Fifth Generation (5G) mobile networks [103], wireless networks [104] and optical networks [105]. However, SDN fault-tolerance is still in its infancy, and there is a broad spectrum of opportunities for the research community to develop new fault-tolerance mechanisms, standards, monitoring, debugging and testing tools to enforce fault-tolerance in such dynamic networking environments, able to ensure Carrier-grade reliability.

LIST OF ABBREVIATIONS/ACRONYMS

ACLs	Access Control Lists
AFRO	Automatic Failure Recovery
APIs	Application Programmable Interfaces
BFD	Bidirectional Forwarding Detection
BFT	Byzantine Fault-tolerant
BLR	Backward Local Rerouting
CPR	Broadband Forum
DFRS	Business Support System
DoS	Denial of Service
EIGRP	Enhanced Interior Gateway Routing Protocol
FIB	Forwarding Information Base
FLR	Forward Local Rerouting
FML	Flow-based Management Language
ForCES	orwarding and Control Element Separation
HAS	Header Space Analysis
LOS	Loss of Signal
MAC	Medium Access Control
NETCONF	Network Configuration Protocol
ONF	Open Network Foundation
OSPF	Open Shortest Path First
P4	Programming Protocol Independent Packet Processors
POF	Protocol-Oblivious Forwarding
REST	REpresentaional State Transfer
RSM	Replicated State Machine
SDN	Software-defined Networking
SMR	State Machine Replication
STS	SDN Troubleshooting System
TCAM	Ternary Content Addressable Memory
VLAN	Virtual Local Area Network
WAN	Wide Area Network
XMPP	Extensible Messaging and Presence Protocol

REFERENCES

[1] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 3rd Quart., 2014.

[2] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[3] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Commun. Surv. Tuts.*, vol. 16, no. 4, pp. 2181–2206, Nov. 2014.

[4] T. Bakhshi, "State of the art and recent research advances in software defined networking," *Wireless Commun. Mobile Comput.*, Jan. 2017, Art. no. 7191647. doi: 10.1155/2017/7191647.

[5] G. Stanley. (2019). *Fault Management—The Overall Process and Life Cycle of a Fault*, Accessed: Dec. 12, 2019. [Online]. Available: https://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/Fault-Management/fault-management.htm

[6] K. Nørvåg, "An introduction to fault-tolerant systems," Dept. Comput. Inf. Sci., Norwegian Univ. Sci. Technol., Trondheim, Norway, Tech. Rep. 6/99, 2000.

[7] A. Lara, A. Kolasani, and B. Ramamurthy, "Simplifying network management using software defined networking and OpenFlow," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS)*, Dec. 2012, pp. 24–29.

[8] P. C. da Rocha Fonseca and E. S. Mota, "A survey on fault management in software-defined networks," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2284–2321, 4th Quart., 2017.

[9] Y. Yu, X. Li, X. Leng, L. Song, K. Bu, Y. Chen, J. Yang, L. Zhang, K. Cheng, and X. Xiao, "Fault management in software-defined networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 349–392, 1st Quart., 2018.

[10] M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2017.

[11] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Depend. Sec. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[12] S. Hukerikar and C. Engelmann, "Resilience design patterns—A structured approach to resilience at extreme scale (version 1.0)," 2016, *arXiv:1611.02717*. [Online]. Available: https://arxiv.org/abs/1611.02717

[13] J. H. Saltzer and M. F. Kaashoek, *Principles of Computer System Design: An Introduction*. San Mateo, CA, USA: Morgan Kaufmann, 2009.

[14] R. Jhawar and V. Piuri, "Fault tolerance and resilience in cloud computing environments," in *Computer and Information Security Handbook*, J. R. Vacca, Ed., 3rd ed. Boston, MA, USA: Morgan Kaufmann, 2017, ch. 9, pp. 165–181. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128038437000090

[15] M. Hasan and M. S. Goraya, "Fault tolerance in cloud computing environment: A systematic survey," *Comput. Ind.*, vol. 99, pp. 156–172, Aug. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361517304438

[16] (2018). *Fault Tolerant Software Systems: Techniques (Part 4a)*. [Online]. Available: https://slideplayer.com/slide/10093304/[Accessed: 12-Dec-2018]

[17] A. Bucchiarone, H. Muccini, and P. Pelliccione, "Architecting fault-tolerant component-based systems: From requirements to testing," *Electron. Notes Theor. Comput. Sci.*, vol. 168, pp. 77–90, Feb. 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571066107000291

[18] J. Dobson, I. Sommerville, and G. Dewsbury, *Introduction: Dependability and Responsibility in Context*. London, U.K.: Springer, 2007, pp. 1–17. doi: 10.1007/978-1-84628-626-1_1.

[19] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, *Software-Defined Networking (SDN): Layers and Architecture Terminology*, document RFC 7426, Internet Requests for Comments, RFC Editor, Jan. 2015. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc7426.txt.pdf

[20] G. Warnock and A. Nathoo, *Alcatel-Lucent Network Routing Specialist II (NRS II) Self-Study Guide: Preparing for the NRS II Certification Exams*. Hoboken, NJ, USA: Wiley, 2011.

[21] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[22] O. Blial, M. Ben Mamoun, and R. Benaini, "An overview on SDN architectures with multiple controllers," *J. Comput. Netw. Commun.*, Apr. 2016, Art. no. 9396525. doi: 10.1155/2016/9396525.

[23] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, "Resilience support in software-defined networking: A survey," *Comput. Netw.*, vol. 92, pp. 189–207, Dec. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615003229

[24] W. Braun and M. Menth, "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014. [Online]. Available: https://www.mdpi.com/1999-5903/6/2/302

[25] E. Haleplidis, J. H. Salim, J. M. Halpern, S. Hares, K. Pentikousis, K. Ogawa, W. Wang, S. Denazis, and O. Koufopavlou, "Network programmability with ForCES," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1423–1440, 3rd Quart., 2015.

[26] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, *Network Configuration Protocol (NETCONF)*, document 6241, IETF, 2011.

[27] P. Saint-Andre, *Extensible Messaging and Presence Protocol (XMPP): Core*, document RFC 2779, IETF, 2011.

[28] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: A survey," *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 24–31, Nov. 2013.

[29] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, 1st ed. Reading, MA, USA: Addison-Wesley, 2015, pp. 80–85.

[30] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements," *Comput. Commun.*, vol. 36, no. 6, pp. 656–665, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366412003349

[31] J. Chen, J. Chen, F. Xu, M. Yin, and W. Zhang, "When software defined networks meet fault tolerance: A survey," in *Algorithms and Architectures for Parallel Processing*, G. Wang, A. Zomaya, G. Martinez, and K. Li, Eds. Cham, Switzerland: Springer, 2015, pp. 351–368.

[32] D. Katz and D. Ward, *Bidirectional Forwarding Detection (BFD)*, document RFC 5880, Internet Requests for Comments, RFC Editor, Jun. 2010. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc5880.txt.pdf

[33] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *Proc. 18th IEEE Workshop Local Metropolitan Area Netw. (LANMAN)*, Oct. 2011, pp. 1–6.

[34] A. U. Rehman, R. L. Aguiar, and J. P. B. Barraca, "A proposal for fault-tolerant and self-healing hybrid SDN control network," in *Proc. 9th Simpósio de Informática (INForum)*, Oct. 2017, pp. 47–52.

[35] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, 2013. doi: 10.1145/2534169.2486012.

[36] C. Trois, M. D. Del Fabro, L. C. E. de Bona, and M. Martinello, "A survey on SDN programming languages: Toward a taxonomy," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2687–2712, 4th Quart., 2016.

[37] G. N. Nde and R. Khondoker, "SDN testing and debugging tools: A survey," in *Proc. 5th Int. Conf. Informat., Electron. Vis. (ICIEV)*, May 2016, pp. 631–635.

[38] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with pyretic," in *Proc. USENIX*, 2013, pp. 1–7.

[39] R. Beckett, X. K. Zou, S. Zhang, S. Malik, J. Rexford, and D. Walker, "An assertion language for debugging SDN applications," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*. New York, NY, USA: ACM, 2014, pp. 91–96. doi: 10.1145/2620728.2620743.

[40] N. L. Van Adrichem, B. J. Van Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *Proc. IEEE EWSDN*, Sep. 2014, pp. 61–66.

[41] L. Sidki, Y. Ben-Shimol, and A. Sadovski, "Fault tolerant mechanisms for SDN controllers," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 173–178.

[42] K. Kuroki, N. Matsumoto, and M. Hayashi, "Scalable OpenFlow controller redundancy tackling local and global recoveries," in *Proc. 5th Int. Conf. Adv. Future Internet*, Barcelona, Spain, 2013, pp. 25–31.

[43] Y. Tingting, H. Xiaohong, M. Maode, and Y. Jie, "Balance-based SDN controller placement and assignment with minimum weight matching," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[44] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The controller placement problem in software defined networking: A survey," *IEEE Netw.*, vol. 31, no. 5, pp. 21–27, Sep./Oct. 2017.

[45] Y. Jiménez, C. Cervelló-Pastor, and A. J. Garcia, "On the controller placement for designing a distributed SDN control layer," in *Proc. IEEE Netw. Conf. (IFIP)*, Jun. 2014, pp. 1–9.

[46] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proc. 9th USENIX Conf. Operating Syst. Design Implement. (OSDI)*. Berkeley, CA, USA: USENIX Association, 2010, pp. 351–364. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924968

[47] A. S. Muqaddas, P. Giaccone, A. Bianco, and G. Maier, "Inter-controller traffic to support consistency in ONOS clusters," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 4, pp. 1018–1031, Dec. 2017.

[48] V. Pashkov, A. Shalimov, and R. Smeliansky, "Controller failover for SDN enterprise networks," in *Proc. Int. Sci. Technol. Conf., Modern Netw. Technol. (MoNeTeC)*, Oct. 2014, pp. 1–6.

[49] F. Németh, R. Steinert, P. Kreuger, and P. Sköldström, "Roles of DevOps tools in an automated, dynamic service creation architecture," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2015, pp. 1153–1154.

[50] N. Handigol, B. Heller, V. Jeyakumar, and D. Maziéres, and N. McKeown, "Where is the debugger for my software-defined network?" in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*. New York, NY, USA: ACM, 2012, pp. 55–60. doi: 10.1145/2342441.2342453.

[51] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*. Berkeley, CA, USA: USENIX Association, 2013, pp. 99–112. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482638

[52] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Fast failure recovery for in-band OpenFlow networks," in *Proc. 9th Int. Conf. Design Reliable Commun. Netw. (DRCN)*, Mar. 2013, pp. 52–59.

[53] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "In-band control, queuing, and failure recovery functionalities for OpenFlow," *IEEE Netw.*, vol. 30, no. 1, pp. 106–112, Jan./Feb. 2016.

[54] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Fault tolerance in TCAM-limited software defined networks," *Comput. Netw.*, vol. 116, pp. 47–62, Apr. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128617300476

[55] H. Li, Q. Li, Y. Jiang, T. Zhang, and L. Wang, "A declarative failure recovery system in software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[56] M. Kuzniar, P. Perešíni, N. Vasić, M. Canini, and D. Kostić, "Automatic failure recovery for software-defined networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*. New York, NY, USA: ACM, 2013, pp. 159–160. doi: 10.1145/2491185.2491218.

[57] H. Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner, and N. Feamster, "CORONET: Fault tolerance for software defined networks," in *Proc. 20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct./Nov. 2012, pp. 1–2.

[58] L. Schiff, S. Schmid, and M. Canini, "Ground control to major faults: Towards a fault tolerant and adaptive SDN control network," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshop (DSN-W)*, Jun./Jul. 2016, pp. 90–96.

[59] J. Chen, J. Chen, J. Ling, and W. Zhang, "Failure recovery using vlan-tag in SDN: High speed with low memory requirement," in *Proc. IEEE 35th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Dec. 2016, pp. 1–9.

[60] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[61] P. Thorat, S. M. Raza, D. T. Nguyen, G. Im, H. Choo, and D. S. Kim, "Optimized self-healing framework for software defined networks," in *Proc. 9th Int. Conf. Ubiquitous Inf. Manage. Commun. (IMCOM)*. New York, NY, USA: ACM, 2015, pp. 7:1–7:6. doi: 10.1145/2701126.2701235.

[62] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res. (SOSR)*. New York, NY, USA: ACM, 2015, pp. 4:1–4:12. doi: 10.1145/2774993.2774996.

[63] F. A. Botelho, F. M. V. Ramos, D. Kreutz, and A. N. Bessani, "On the feasibility of a consistent and fault-tolerant data store for SDNs," in *Proc. 2nd Eur. Workshop Softw. Defined Netw. (EWSDN)*. Washington, DC, USA, Oct. 2013, pp. 38–43. doi: 10.1109/EWSDN.2013.13.

[64] F. Botelho, A. Bessani, F. M. V. Ramos, and P. Ferreira, "On the design of practical fault-tolerant SDN controllers," in *Proc. 3rd Eur. Workshop Softw. Defined Netw. (EWSDN)*, Sep. 2014, pp. 73–78.

[65] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A replication component for resilient OpenFlow-based networking," in *Proc. IEEE Netw. Oper. Manage. Symp.*, Apr. 2012, pp. 933–939.

[66] W. H. F. Aly and Y. Kotb, "Towards SDN fault tolerance using Petri-nets," in *Proc. 28th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2018, pp. 1–3.

[67] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw. (INM/WREN)*. Berkeley, CA, USA: USENIX Association, 2010, p. 3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.1863136

[68] A. J. Gonzalez, G. Nencioni, B. E. Helvik, and A. Kamisinski, "A fault-tolerant and consistent SDN controller," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[69] K. ElDefrawy and T. Kaczmarek, "Byzantine fault tolerant software-defined networking (SDN) controllers," in *Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, Jun. 2016, pp. 208–213.

[70] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 136–141, Feb. 2013.

[71] A. Bessani, J. Sousa, and E. E. P. Alchieri, "State machine replication for the masses with BFT-SMaRt," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2014, pp. 355–362.

[72] B. Heller, C. Scott, N. McKeown, S. Shenker, A. Wundsam, H. Zeng, S. Whitlock, V. Jeyakumar, N. Handigol, J. McCauley, K. Zarifis, and P. Kazemian, "Leveraging SDN layering to systematically troubleshoot networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*. New York, NY, USA: ACM, 2013, pp. 37–42. doi: 10.1145/2491185.2491197.

[73] C. Scott, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai, E. Huang, Z. Liu, A. El-Hassany, S. Whitlock, H. B. Acharya, K. Zarifis, and S. Shenker, "Troubleshooting blackbox SDN control software with minimal causal sequences," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 395–406, 2014. doi: 10.1145/2740070.2626304.

[74] M. Canini, D. Venzano, and P. Perešíni, D. Kostić, and J. Rexford, "A NICE way to test OpenFlow applications," presented at the 9th USENIX Symp. Netw. Syst. Design Implement. (NSDI), 2012, pp. 127–140.

[75] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "FatTire: Declarative fault tolerance for software-defined networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*. New York, NY, USA: ACM, 2013, pp. 109–114. doi: 10.1145/2491185.2491187.

[76] B. Chandrasekaran and T. Benson, "Tolerating SDN application failures with LegoSDN," in *Proc. 13th ACM Workshop Hot Topics Netw. (HotNets-XIII)*. New York, NY, USA: ACM, 2014, pp. 22:1–22:7. doi: 10.1145/2670518.2673880.

[77] B. Chandrasekaran, B. Tschaen, and T. Benson, "Isolating and tolerating SDN application failures with LegoSDN," in *Proc. Symp. SDN Res. (SOSR)*. New York, NY, USA: ACM, 2016, pp. 7:1–7:12. doi: 10.1145/2890955.2890965.

[78] Open Network Foundation. *Open Network Operating System (ONOS)*. Accessed: Jun. 20, 2019. [Online]. Available: https://www.opennetworking.org/onos/

[79] *The POX Network Software Platform*. Accessed: Jun. 20, 2019. [Online]. Available: https://github.com/noxrepo/pox

[80] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and Scott Shenker, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008. doi: 10.1145/1384609.1384625.

[81] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *Proc. 16th ACM SIGPLAN Int. Conf. Funct. Program. (ICFP)*. New York, NY, USA: ACM, 2011, pp. 279–291. doi: 10.1145/2034773.2034812.

[82] *Project Floodlight*. Accessed: Jun. 20, 2019. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[83] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 527–538, Aug. 2014. doi: 10.1145/2740070.2626314.

[84] M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford, "Network architecture for joint failure recovery and traffic engineering," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Modeling Comput. Syst. (SIGMETRICS)*. New York, NY, USA: ACM, 2011, pp. 97–108. doi: 10.1145/1993744.1993756.

[85] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proc. 1st ACM Workshop Res. Enterprise Netw. (WREN)*. New York, NY, USA: ACM, 2009, pp. 1–10. doi: 10.1145/1592681.1592683.

[86] S. Lowe, J. Edelman, and M. Oswalt, *Network Programmability and Automation*, 1st ed. Champaign, IL, USA: O'Reilly Media, 2017, pp. 1–35.

[87] D. F. Macedo, D. Guedes, L. F. M. Vieira, M. A. M. Vieira, and M. Nogueira, "Programmable Networks—From Software-Defined Radio to Software-Defined Networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 1102–1125, 2nd Quart., 2015.

[88] X. Foukas, M. K. Marina, and K. Kontovasilis, "Software defined networking concepts," in *Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture*. Hoboken, NJ, USA: Wiley, 2015, ch. 3, pp. 21–44. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118900253.ch3

[89] A. U. Rehman, R. L. Aguiar, and J. P. Barraca, "Network functions virtualization: The long road to commercial deployments," *IEEE Access*, vol. 7, pp. 60439–60464, 2019.

[90] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014. doi: 10.1145/2656877.2656890.

[91] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*. New York, NY, USA: ACM, 2013, pp. 127–132. doi: 10.1145/2491185.2491190.

[92] R. Tischer and J. Gooley, *Programming and Automating Cisco Networks*, 1st ed. Indianapolis, IN, USA: Cisco Press, 2016, pp. 1–64.

[93] A. Galis, S. Clayman, L. Mamatas, J. R. Loyola, A. Manzalini, S. Kuklinski, J. Serrat, and T. Zahariadis, "Softwarization of future networks and services-programmable enabled networks as next generation software defined networks," in *Proc. IEEE SDN Future Netw. Services (SDN4FNS)*, Nov. 2013, pp. 1–7.

[94] P. Perešíni, M. Kuźniar, and D. Kostić, "Monocle: Dynamic, fine-grained data plane monitoring," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*. New York, NY, USA: ACM, 2015, pp. 32:1–32:13. doi: 10.1145/2716281.2836117.

[95] K. Bu, X. Wen, B. Yang, Y. Chen, L. E. Li, and X. Chen, "Is every flow on the right track?: Inspect SDN forwarding with RuleScope," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[96] P. Zhang, H. Li, C. Hu, L. Hu, L. Xiong, R. Wang, Y. Zhang, "Mind the gap: Monitoring the control-data plane consistency in software defined networks," in *Proc. 12th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*. New York, NY, USA: ACM, 2016, pp. 19–33. doi: 10.1145/2999572.2999605.

[97] H. Farhad, H. Lee, and A. Nakao, "Data plane programmability in SDN," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 583–588.

[98] W. L. da Costa Cordeiro, J. A. Marques, and L. P. Gaspary, "Data plane programmability beyond OpenFlow: Opportunities and challenges for network and service operations and management," *J. Netw. Syst. Manage.*, vol. 25, no. 4, pp. 784–818, Oct. 2017. doi: 10.1007/s10922-017-9423-2.

[99] M. Bouet, K. Phemius, and J. Leguay, "Distributed SDN for mission-critical networks," in *Proc. IEEE Mil. Commun. Conf.*, Oct. 2014, pp. 942–948.

[100] V. Gkioulos, H. Gunleifsen, and G. K. Weldehawaryat, "A systematic literature review on military software defined networks," *Future Internet*, vol. 10, no. 9, p. 88, 2018. [Online]. Available: https://www.mdpi.com/1999-5903/10/9/88

[101] R. Carreras Ramirez, Q.-T. Vien, R. Trestian, L. Mostarda, and P. Shah, "Multi-path routing for mission critical applications in software-defined networks," in *Industrial Networks and Intelligent Systems*, T. Q. Duong and N.-S. Vo, Eds. Cham, Switzerland: Springer, 2019, pp. 38–48.

[102] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.*, vol. 112, pp. 279–293, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912861630411X

[103] A. Hakiri and P. Berthou, *Leveraging SDN for the 5G Networks*. Hoboken, NJ, USA: Wiley, 2015, ch. 5, pp. 61–80. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118900253.ch5

[104] I. T. Haque and N. Abu-Ghazaleh, "Wireless software defined networking: A survey and taxonomy," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2713–2737, 4th Quart., 2016.

[105] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, "Software defined optical networks (SDONs): A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2738–2786, 4th Quart., 2016.

A. U. REHMAN received the bachelor's degree (Hons.) in telecommunications engineering from Mohammad Ali Jinnah University, Pakistan, in 2009, and the master's degree (Hons.) in telecommunications engineering from The University of Sunderland, U.K., in 2011. He is currently pursuing the Ph.D. degree in telecommunications with MAP-tele (a joint Doctoral Program of the Universidade do Porto, the Universidade de Aveiro, and the Universidade do Minho, Portugal, all three universities with a strong tradition in the area of telecommunications engineering). He was a Visiting Instructor with Telecom Foundation, Pakistan. He was a Teaching Assistant with Mohammad Ali Jinnah University, for several years. He is currently involved in the research areas of telecommunications and the Internet at the Instituto de Telecomunicações, Portugal, where he is currently an Active Member of the Network Application and Services Group. His research interests include software-defined networking (SDN), network functions virtualization (NFV), and the reliability and resilience of future networks. He is also a member of the Communications Society (ComSoc) and the IEEE Software Defined Networks Community.

RUI. L. AGUIAR received the degree in telecommunication engineering and the Ph.D. degree in electrical engineering from the Universidade de Aveiro, in 1990 and 2001, respectively, where he is currently a Full Professor and is responsible for networking area. He has been an Adjunct Professor with INI, Carnegie Mellon University, and a Visiting Research Scholar with the Universidade Federal de Uberlândia, Brazil. He is coordinating a research line in the area of networks and multimedia nationwide with the Instituto de Telecomunicações. His current research interests include the implementation of 5G networks and the future Internet. He has over 450 published articles in his research areas, including standardization contributions to the IEEE and the IETF. He is also a Senior Member of the Portugal ComSoc Chapter Chair and a member of the ACM. He has served as the Technical and General Chair of several IEEE, ACM, and IFIP conferences and as an IEEE ComSoc Distinguished Lecturer. He is the current Chair of the Steering Board of the Networld 2020 ETP. He is regularly invited for keynotes on 5G and the future Internet subjects. He sits on the TPC of most major IEEE ComSoc conferences. He is also an Associate Editor of ETT (Wiley) and *Wireless Networks* (Springer). He has helped in the launch of *ICT Express* (Elsevier).

JOÃO PAULO BARRACA received the Ph.D. degree in informatics engineering from the Universidade de Aveiro, in 2012, where he is currently an acting Assistant Professor. He conducts research with the Instituto de Telecomunicações, having led the TN-AV Group, from 2015 to 2016. He has close to 100 peer-reviewed publications and reports related to solutions for the Internet of Things and software for cloud environments, with a focus on software-defined networking and 5G Networks. Having participated in many review panels, he has also organized workshops and conferences. He has participated in more than 20 projects, either developing novel concepts or applying these concepts in innovative products and solutions. He leads the FCT/CAPES DEVNF Project in Portugal devoted to NFV orchestration, the local teams of EU LIFE-PAYT, participates in European Science Cloud for Astronomy (EU AENEAS), the local team in the P2020 (CRUISE Project), the security team at P2020-Social, participates in the EU Interreg CISMOB smart cities pilot, the Engage SKA research infrastructure, and the Square Kilometer Array System (SKA) Team, and having lead activities for TM-LINFRA, among a dozen other innovation projects. Recently, he received the third place from the INCM Innovation Challenge, for the development of a project targeting smarter environments for public transports in smart cities, using blockchain technologies.