ARTICLE IN PRESS

Digital Communications and Networks xxx (xxxx) xxx

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

Self-healing and SDN: bridging the gap

Leonardo Ochoa-Aday [*], Cristina Cervelló-Pastor, Adriana Fernández-Fernández

Department of Network Engineering, Universitat Politècnica de Catalunya (UPC), Esteve Terradas, 7, 08860, Castelldefels, Spain

A R T I C L E I N F O

Keywords:
Software-defined network
Autonomic network management
Protocol design
Fault tolerance
Network management

A B S T R A C T

Achieving high programmability has become an essential aim of network research due to the ever-increasing internet traffic. Software-Defined Network (SDN) is an emerging architecture aimed to address this need. However, maintaining accurate knowledge of the network after a failure is one of the largest challenges in the SDN. Motivated by this reality, this paper focuses on the use of self-healing properties to boost the SDN robustness. This approach, unlike traditional schemes, is not based on proactively configuring multiple (and memory-intensive) backup paths in each switch or performing a reactive and time-consuming routing computation at the controller level. Instead, the control paths are quickly recovered by local switch actions and subsequently optimized by global controller knowledge. Obtained results show that the proposed approach recovers the control topology effectively in terms of time and message load over a wide range of generated networks. Consequently, scalability issues of traditional fault recovery strategies are avoided.

1. Introduction

The drastic increase of Internet services, such as video on demand, big data, server virtualization and cloud services, is one of the trends driving the network industry to change its traditional network architectures. To meet the ever-increasing demands of new services, network operators require emerging solutions to effectively manage their network resources using more flexible and dynamic schemes.

In this context, Software-Defined Network (SDN) has emerged as a promising approach for managing complex and heterogeneous network infrastructures [1]. In essence, this new paradigm proposes decoupling the control plane from the forwarding plane by centralizing intelligence, state of the network and control functions in an entity called the controller [2]. In spite of the logically centralized control in SDNs, the control plane can be implemented using multiple physically-distributed servers in order to mitigate the problems of scalability and reliability [3].

However, the widespread adoption of the SDN in heterogeneous and failure-prone deployments (e.g., data centers, clouds, etc.), is putting increasing pressure on the survivability strategies of the control plane to guarantee the plane's resilience at all times [4]. In fact, improving the reliability of the SDN has been identified as one of the next crucial objectives for research and industry efforts, and this becomes more challenging when in-band implementations are also considered [5,6]. Moreover, an adequate solution should also address scalability concerns and allow for rapid responses to network events and requirements that can be met by eliminating the controller intervention in the failure recovery procedure. To address these issues, there has been growing interest in combining autonomic principles of self-healing with the SDN to develop resilient programmable networks.

In 2001, the concept of Autonomic Computing (AC) was introduced by Paul Horn to the National Academy of Engineers at Harvard University [7]. This idea was inspired by the autonomic operation of the nervous system in the human body, which is able to make independent choices to modify its behavior in the face of different stimuli [8]. This ability to make independent choices defines an autonomic entity and this definition can be applied to multiple contexts.

In terms of computer networks, this paradigm has been translated into the Autonomic Network Management (ANM) [9]. The main idea underlying this proposal is the leveraging of self-management properties (i.e., self-configuration, self-healing, self-optimization and self-protection) in complex scenarios as heterogeneous network environments [10]. The four aforementioned properties, referred to in the existing literature as the "self-CHOP properties," collectively define an autonomic system and have been attracting growing attention from both academia and industry.

The ANM aims to enable networks to work in a completely unsupervised manner. In essence, autonomic networks should be capable of adapting their behaviors dynamically to meet the specific, changing

* Corresponding author.
E-mail addresses: leonardo.ochoa@entel.upc.edu (L. Ochoa-Aday), cristina@entel.upc.edu (C. Cervelló-Pastor), adriana.fernandez@entel.upc.edu (A. Fernández-Fernández).

https://doi.org/10.1016/j.dcan.2019.08.008
Received 15 November 2018; Received in revised form 5 July 2019; Accepted 30 August 2019
Available online xxxx

needs of individual users and high-level application goals [11]. Moreover, the ANM seeks to dramatically decrease the complexity and costs associated with reliable deployments of network and communication services [12,13].

The general framework defined in the ANM follows a similar logic as the SDN architecture. The framework is composed of a set of managed elements distributed in the network, which are centrally governed by an autonomic manager [14]. The former group (i.e., managed elements) is intended to serve as an interface with the system, while the latter (i.e., autonomic manager) manages the different operations of the components. In essence, the autonomic manager is responsible for performing the required adaptations in order to achieve a set of higher-level goals. To accomplish this, a global view and knowledge regarding the managed entities and their management operations are required by the centralized entity.

Besides using its holistic knowledge, the autonomic manager needs to perform four main tasks, namely, monitoring the managed entities, analyzing their performance, planning appropriate management operations and executing them [15]. These functionalities, also known as the Monitor, Analyze, Plan and Execute (MAPE) loop, are of paramount importance to enhancing the network performance and obtaining solutions to current or anticipated problems. Another approach to attain the autonomic principles is to distribute these functionalities over the set of managed components, which interact with each other to provide convergent autonomic management, enabling the self-adaptation to the environment changes.

In this way, the ANM enables autonomous real-time management of network infrastructures and replaces traditional manual and semi-automatic managing approaches, which are already costly and time-consuming. Besides sharing a common objective, i.e., enabling real-time programmable, self-adaptable and cost-effective networks and services, the SDN and the ANM can complement each other to provide high-level operator objectives, such as enhanced fault-tolerance, cyber-attack mitigation and performance guarantees [16,17].

The aim of this work is, therefore, to provide an inherent fault-recovery mechanism for the survivability of the control plane, while maintaining an accurate global network view at the controller through autonomic principles applied in self-healing SDN environments. Specifically, we seek to exploit self-healing properties in order to reduce the global time, as well as the number of control messages required to recover the control plane connectivity from a network failure. To do so, in this paper we shape the design of a novel Self-Healing Protocol (SHP) to boost the control plane resilience in the SDN-managed environments without overburdening the SDN controller. In this regard, we leverage forwarding devices with autonomic attributes in order to recover the network from failures in an autonomous and stable fashion by only taking actions at the switch level.

The remainder of this paper is organized as follows: In Section 2 we provide an overview of the self-healing property and failure management reality in the SDN. After this, a brief description of some proposals in existing literature related with fault management in the SDN is outlined in Section 3. In Section 4, we define the considered architecture and present the proposed framework. Then, in Section 5 we describe in detail the autonomic fault recovery mechanism. The performance of the proposed solution assessed using several evaluation metrics is analyzed in Section 6 through experimental simulations using the Objective Modular Network Testbed in C++ (OMNeT++). Finally, we draw some conclusions in Section 7.

2. Self-healing SDN environments

As previously mentioned, self-healing is one of the four main properties conceived in the autonomic paradigm [18,19]. This term refers to the capability of the network to restore its operations, independently and without external intervention, when any failure occurs. Every system with self-healing properties has the capability to discover, diagnose and react to failures. The primary objective of integrating self-healing features into any network operation is improving its reliability and maintainability. These quality attributes are traditionally heightened in self-healing systems [20,21].

As failure management is a critical aspect for every network operation, substantial efforts have been devoted to the implementation of different strategies. Traditional failure recovery strategies are classified into two groups: restoration and protection [22]. The former strategy is reactive and requires the online computation and dynamic installation of alternate routes after the detection of a failure. In the protection strategy, however, backup paths are pro-actively configured across the network. Therefore, while the restoration scheme requires longer recovery time, the protection approach imposes higher memory requirements and raises scalability concerns.

The SDN provides flexibility to network systems and increases opportunities for innovation and development, but there is no guarantee that these networks are robust [23]. In fact, implementing crucial functions in SDN environments, such as failure resilience, is a highly complex task, given that the controller intervention implies non-negligible delays and signaling overheads (due to the propagation delay of failure notifications and reactive failure recovery countermeasures). Moreover, failures in the control plane will have a significant impact on the network performance, since these failures may cause the problem that new flow entries cannot be handled promptly. Therefore, failure resilience is clearly one determinant requirement that must be addressed for the successful adoption of SDN technology.

OpenFlow-based fault recovery approaches have been mainly focused on restoring failed data paths by locally detouring individual flows [24–26]. These mechanisms, also referred to as fast-failover techniques, avoid the controller intervention during recovery, reducing the incurred recovery time. Although local reactions to failures are faster than path-based end-to-end reallocations, this scheme has some crucial drawbacks. First, it can be used only if alternative path rules are available at the node that detects the failure. Moreover, it requires instantiating multiple alternate path rules for each flow entry on each link, which implies an inefficient resource allocation and may be impractical in some cases. Lastly, in large topologies, an extensive computation of a backup alternative for each flow passing through each node may overload the centralized controller and create a processing bottleneck.

A reliable and scalable mechanism to recover a link or node failure has additional requirements in the context of in-band SDN. With in-band control, an additional physical control network is not needed since the control traffic is sent with the data traffic over the same infrastructure [27]. In such scenarios, failures in the interconnection between forwarding devices are likely to also affect the control plane. In fact, it is highly possible that a failure in a link or node will disconnect several switches from the controller, making the recovery task much more complicated. Therefore, adequate solutions must not only try to recover the control plane connectivity within the shortest possible time, but also be able to achieve this even when the controller is unreachable.

In this work, the term “self-healing SDN environment” is used to refer to a system that proactively monitors its service parameters and network elements in different segments in order to recover from errors after a failure has been detected [28]. In essence, it allows reactions to network component (i.e., links or nodes) failures by reconfiguring traffic allocation in order to make use of the surviving network infrastructure able to provide services. Moreover, based on the internal information about appropriate metrics, the system can forecast future service failures and propose preventative actions before the service fails. In this way, it is possible to avoid any outage of essential services, such as the network topology discovery.

3. Fault management in SDN: a literature review

In this section, we first discuss related works published in the area of fault management in the SDN by improving the standard OpenFlow solution. Subsequently, we analyze some research efforts focusing on leveraging self-healing frameworks in the SDN.

3.1. OpenFlow-based frameworks

Improving the robustness of the SDN has been identified as one of the most important tasks to be addressed by research on the SDN [29]. However, it has thus far been one of the least researched topics.

Capone et al. [30,31] proposed a fast and reliable detour plan for failure management in the SDN. Their framework relies on OpenState, an OpenFlow extension that enables switches to perform match-action rules depending on states triggered by packet-level events. In this way, the control logic of the SDN controllers related to failure management is in part offloaded onto the forwarding devices. Simulation results show the suitability of this approach compared with a classic end-to-end path protection scheme and with respect to an approach based on the Open-Flow fast-failover mechanism [24]. However, in Ref. [30], the authors use optimization models for the computation of backup paths, while [31] requires an initial provisioning of a backup forwarding policy in every switch. Our proposal, on the other hand, is based on local switch actions to quickly recover the control paths, being more efficient and scalable.

Table 1
Comparison between the proposed approach and other state-of-the-art solutions.

Mechanism	Main Features	Recovery strategy
[30,31]	Protection schemes based on the use of precomputed backup paths and the OpenState extension to react to packet-level events and extended with a probing scheme to establish if the original failure has been resolved.	Protection
[32,33]	Local detouring mechanisms that rely on flow grouping and aggregation methods for failure handling in OpenFlow networks while addressing the issue of flow table space constraints for preconfiguration of alternate paths.	Protection
[34]	Failover scheme that uses preconfigured primary and secondary paths computed by an OpenFlow controller and set on every switch in terms of fast-failover rules using per-link, BFD sessions to quickly detect link failures.	Protection
[36]	Active probing technique to detect and manage failures in an OpenFlow based data center network exploiting load balancing among equal cost multiple paths without involving the controller in order to avoid scalability issues and achieve faster recovery.	Restoration
[37]	Optimized self-healing SDN framework, which includes a rapid recovery (RR) mechanism to perform an immediate link recovery at the switch level and an optimal alternate path computation after recovering from a failure.	Restoration
[38]	Fault management framework based on self-healing for 5G networks to ensure the resiliency and availability of end-to-end services in NFV-based architectures that rely on centralized SDN out-of-band and in-band networks.	Restoration
[39,40]	Generic self-healing approach for centralized SDN infrastructures that includes a Bayesian network-based algorithm to detect disruptions on the application plane, the control plane and the data plane at run-time.	Restoration
[41]	Self-healing protocol for automatic discovery and maintenance of the network topology in the SDN that integrates two enhanced features: layer two topology discovery and autonomic fault recovery in a unified mechanism.	Restoration
Current paper	Self-healing mechanism that enables real-time recovery of the control plane connectivity in SDN-managed environments in the face of failures without overburdening the controller performance and suitable for SDN scenarios with in-band control.	Restoration

Enhanced local detouring mechanisms, with flow grouping and aggregation methods for rapid and lightweight failure handling in Open-Flow networks, are also investigated in Refs. [32,33]. Based on the flow grouping strategy, the authors proposed the Controller Independent Proactive (CIP) and ontroller Dependent Proactive (CDP) recovery schemes. Through the performance evaluation in Ref. [32], it was identified that the proposed recovery schemes achieve a 99% reduction in flow storage for an alternate path setup using Virtual Local Area Network (VLAN) tagging and reduce the failure recovery time up to 4 *ms* and 20 *ms* respectively, satisfying the 50 *ms* total failure recovery time required in carrier networks. However, these approaches are tied to the actual implementation of the optional fast-failover group feature of OpenFlow, which limits their applicability.

Similarly, a fast (i.e., sub 50 *ms*) failover scheme was introduced in Ref. [34]. This scheme relies on the link-failure detection by combining the primary and backup paths configured by a central OpenFlow controller. Moreover, the authors implemented a per-link failure detection using Bidirectional Forwarding Detection (BFD) [35], a protocol that identifies failures by detecting packet loss in frequent streams of control messages. Performance measurements in a hardware switch OpenFlow-based testbed show that the recovery time of sub 50 *ms* can be achieved by configuring the BFD transmit interval at 15 *ms*. The faster recovery time of 3.3 *ms* is obtained after further decreasing the BFD interval to 1 *ms*. The experimental evaluation confirms that the recovery time achieved are independent of the path length and the network size. Unlike our proposal, this solution also depends on the fast-failover group support, which may vary across different switch implementations.

In Ref. [36], the authors studied the impact of network failures on the deployment of load balancing mechanisms in data center networks based on the OpenFlow protocol. They used an active probing method to detect and manage failures, exploiting the load balancing among equal cost multiple paths. By exploiting this technique, all of Top-of-Rack (ToR) switches can perform local configuration modifications and act independently of the central controller, avoiding the saturation and scalability issues of the SDN controllers. However, the proposed strategy is limited to tree-like topologies and is incompatible with environments without a dedicated out-of-band control network.

Although the aforementioned proposals [30–34,36] have eliminated the drawbacks of SDN controller interventions (in terms of packages overhead and control latency) by taking actions at the switch level only, these solutions are limited to recovering the system from failures and do not consider the performance guarantees after restoring the network.

3.2. Self-healing frameworks

The widespread adoption of the SDN in heterogeneous and failure-prone deployments (i.e., data centers and clouds) has raised a general interest in providing the SDN with the self-healing paradigm [9,19,21]. In relation to this, some researchers have proposed novel frameworks to improve the resiliency and predictability of SDNs.

Thorat et al. [37] proposed a self-healing SDN framework that optimizes recovery by applying autonomic principles. The proposed framework includes a Rapid Recovery (RR) mechanism on the switch level and an Optimized Self-Healing (OSH) module on the control plane. After a failure occurs, the RR mechanism must recover the network connectivity as soon as possible to minimize service disruption time. Then, the OSH module uses the network information to calculate new optimal paths. Based on the analytical model proposed, the authors proved a reduction in backup flow entries after a failure of 99% per switch. Although the RR mechanism exploits the efficiency of link protection schemes, it requires the OpenFlow group table feature to be implemented.

The vulnerabilities of the SDN and etwork Function Virtualization (NFV) are also analyzed in Ref. [38] from a fault management perspective. The authors proposed a self-healing-based framework to ensure the resiliency and availability of end-to-end services and resources in 5G networks. This framework interacts with the three planes of the SDN (i.e.,

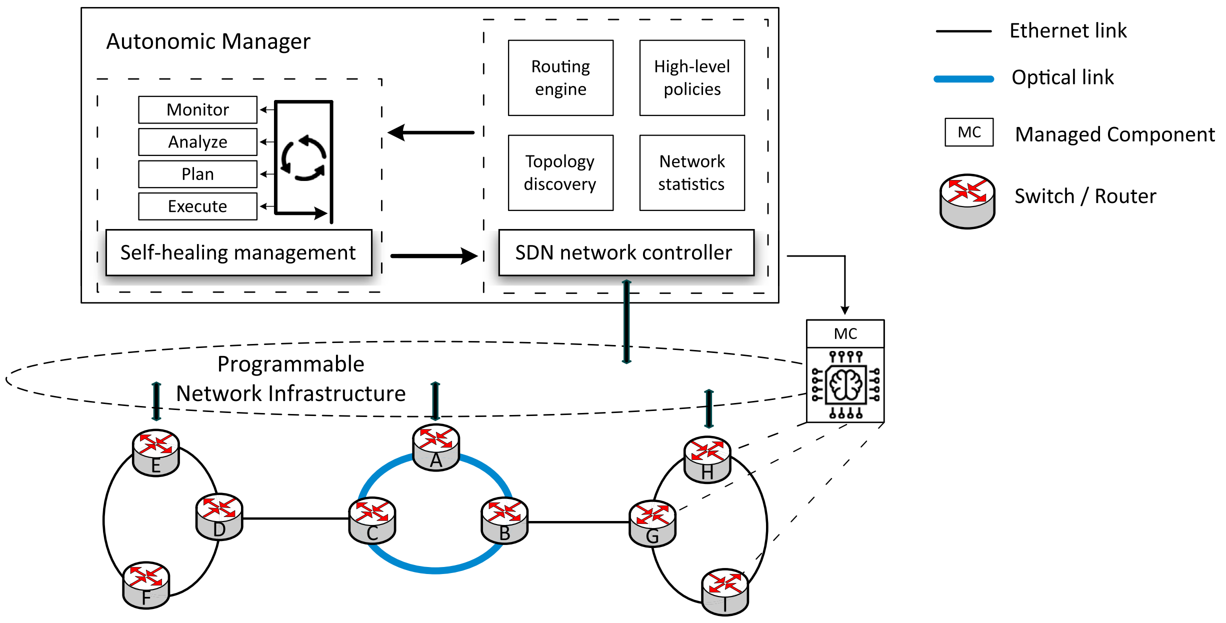


Fig. 1. Overall system architecture for the proposed solution.

the application, control and data planes) by taking observations from the network and launching recovery actions. The proposed self-healing framework for SDN/NFV-based networks defines two types of actions, namely, those that heal the SDN architecture and those that cooperate with the NFV Infrastructure (NFVI) to avoid any service interruption (i.e., proactive actions). The authors translated part of the self-healing framework into a specific SDN platform by proposing a diagnosis block in the control plane. They proved that this module can detect any unavailability of a multicast service as well as reactively resolve malfunctions at several levels.

In Refs. [39,40], the same authors proposed a generic self-healing approach based on Bayesian network models for a diagnosis block. In Ref. [39], the authors developed an algorithm into a self-healing module in a centralized SDN architecture. This infrastructure was emulated on Mininet with POX as the SDN controller. To prove the functionality of the proposed module in the presence of failures, the authors ran a video streaming service delivery through the fixed network topology. Based on this experiment, the authors claim that the self-healing module can detect, diagnose and repair faults of different nature, such as physical failures, streaming services, OpenFlow crashes and drops on any interface [40]. As a result, if the streaming service behaves abnormally, the module detects this, diagnoses the root cause and applies the corresponding actions to reestablish the service.

Despite the potential benefits of the diagnosis block and self-healing modules proposed in [38–40], non-negligible delays and signaling overheads may be required, given that the diagnosis result must be sent to the recovery block, where appropriate strategies to fix the failure are determined. In order to provide a faster recovery, our proposal is based on performing first local actions to reestablish the affected connectivity followed by the subsequent optimization of control paths.

To the best of our knowledge, the closest work to our approach is [41], where a similar distributed operation was briefly introduced to support self-healing properties in the SDN. However, this related work lacks of crucial features for protocol implementation such as message types and dataframes structure. In addition to providing more details about the protocol design, in the present work the recovered control tree can be optimized in terms of delay by the SDN controllers, which have complete knowledge of the network topology and state. Furthermore, a deeper evaluation is performed to analyze the impact of the control traffic on the network due to the self-healing mechanism and the number of nodes involved in the recovery process.

A summary of the discussed recovery mechanisms is presented in Table 1. Each row in the table refers to a different approach. Meanwhile, each column refers to a particular feature: the proposal reference, the description of main features or the employed failure recovery strategy. This paper is also included at the end of the table.

In summary, we believe that there is still room for exploiting the use of self-healing techniques to leverage the reliability and accuracy of the centralized network view and control plane topology in the SDN. Our proposal considers the integration of SDN with autonomic properties in order to provide native fault recovery within the control plane. This is achieved only by taking actions at the switch level, without overburdening the controller, and it is suitable for SDN scenarios with in-band control.

4. Problem statement

Most of current research efforts in the fault management area have been oriented towards proposing recovery mechanisms within the data plane of SDNs. However, a resilient control plane is a critical feature for current SDN deployments. This high-level goal is becoming extremely important due to the growing prevalence of SDNs on large-scale and heterogeneous networks, for which the in-band mode is more practical and cost-efficient. Moreover, achieving robustness in the control plane should not be limited to recovering the system from failures. It should also ensure proper responsiveness regarding performance guarantees (such as control paths delay) once the network is recovered. To that end, exploiting the self-healing property of an ANM system, agreed upon as the next generation of management [42], represents a very suitable approach. Despite this, in our literature review we identified a lack of proposals that integrate cognitive and autonomic management schemes with SDNs.

The question of how much control intelligence should remain in SDN switches remains an issue of ongoing debate [43]. Although our solution embraces the idea of centralized network control decoupled from forwarding devices, we envisage an autonomic SDN environment where

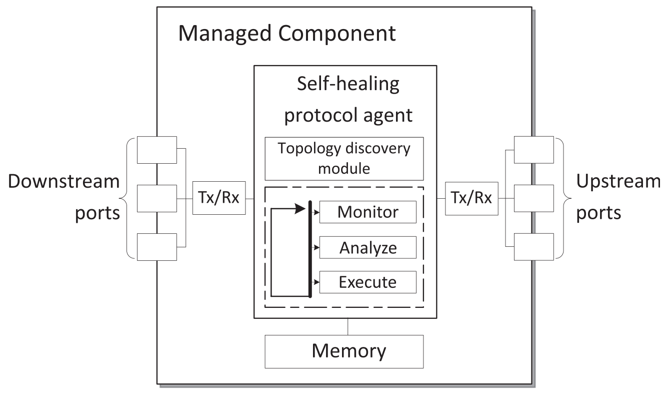


Fig. 2. Schematic diagram of an autonomous forwarding device.

distributed forwarding devices also contribute to providing services like topology discovery and fault recovery.

4.1. *Autonomous system architecture*

By definition, autonomic networks are comprised of two major entities: the managed components and the autonomic manager [9]. The overall system architecture for the proposed solution is shown in Fig. 1. Both elements are identified as follows:

- *Managed components*: These components are represented by the set of forwarding devices that support the proposed SHP. Each managed component includes sensors for monitoring the state of neighboring links and effectors for modifying local parameters in its network.
- *Autonomic manager*: This manager is coupled within each SDN controller, and it has centralized network knowledge and therefore can better diagnose problems. This component is responsible for making tactical decisions and optimizing network performance in order to accomplish high-level objectives (e.g., inherent control plane robustness and minimum-latency control paths).

4.2. *Framework description*

In order to solve the scalability issues of traditional autonomic systems and reduce the time required to recover the control plane in the event of failures, in this approach, the managed components also perform some of the MAPE functionalities. In essence, each forwarding device is equipped with a SHP agent composed of several modules as illustrated in Fig. 2.

The SHP agent allows the forwarding devices to monitor their port interfaces, analyze the collected data and execute the required actions. To do so, techniques like sketches aiming to effectively acquire information about the traffic can be implemented in the forwarding plane [44]. As a result, forwarding devices are capable of autonomously and quickly resolving a link or node failure without the intervention of the controller.

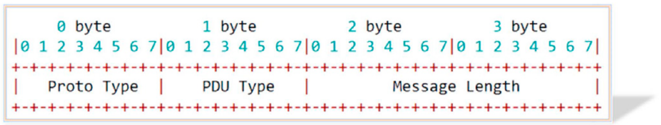
As the topology discovery mechanism is outside the scope of this paper, the SHP inherits the topology discovery module proposed in Ref. [45]. By using this module as a basis, both features (i.e., topology discovery and fault recovery) are integrated into a unified approach for discovering physical topology and providing autonomous fault recovery in the control plane of programmable networks.

In accordance with [45], switches are classified into one of the three possible roles, (i.e., leaf, v-leaf or core). Leaf nodes are the nodes in the network that have only one neighbor. A node is v-leaf when it has more than one neighbor but only one of them can provide a path to the SDN controllers. The remaining switches are denoted as core nodes.

Likewise, ports have different states according to their positions in the control tree formed by the topology discovery module. For the sake of better understanding, the port states defined in Ref. [45] are described below:

Table 2
Functionalities performed by each entity in the autonomic framework.

Task	Managed Components	Autonomic Manager
Monitor	Receive measurement data about the state of neighboring links from sensors residing on the devices.	Collects and consolidates the data obtained from its managed components at different locations.
Analyze	Interpret collected data into a state description according to the incoming port statuses.	Isolates the failure from the wider topology and anticipates further implications using the system knowledge.
Plan	Interact with the neighboring nodes through message exchange to discover alternative control paths.	Optimizes the hierarchical control tree to enhance network performance and achieve a set of higher-level goals.
Execute	Perform local control tree adaptations in order to find an immediate solution to the network failure.	Installs new flow configuration rules in the forwarding devices along optimized control routes.



```
     0 byte          1 byte          2 byte          3 byte
|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Proto Type   |   PDU Type    |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 3. General structure of SHP messages.

- A standby port is an active port in the node that is not used in the control tree.
- A parent port is the upstream port in the control tree. Thus, each node has only one parent port.
- A child port is a downstream port of the control tree.
- A pruned port is a child port that is attached to a leaf or v-leaf node.

A summary of the functionalities performed in the proposed solution by both entities (i.e., managed components and autonomic manager) and classified according to the MAPE tasks, is presented in Table 2.

5. Autonomic SHP

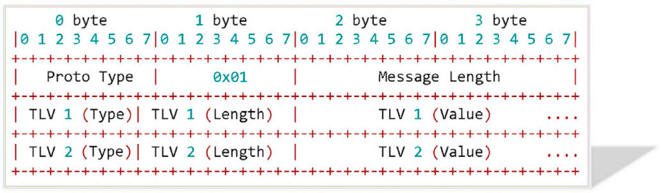
In order to restore the control plane connectivity and maintain an accurate network view in the controller, the proposed fault recovery mechanism is autonomously performed by the SHP components. In particular, this proposal is conceived to provide a quick control plane restoration by only taking local actions while notifying the controller about the network disruption. The controller can then perform an optimized route computation [46].

In this section, we first describe the data frame structure of each message used by the SHP components. Afterward, a detailed description of the protocol operations is provided, including the mathematical formulation used in the control path optimization.

5.1. *Data frames description*

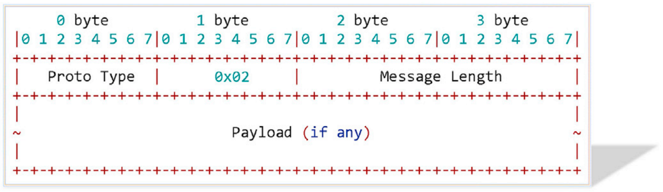
The SHP communications follow the frame encapsulation illustrated in Fig. 3. Accordingly, data frames defined in this proposal use the same header format, where different Protocol Data Unit (PDU) types are included to identify the message.

The fields in this header structure are transmitted from left to right and each tick mark represents a one-bit position in the frame. As illustrated in Fig. 3, the overall header size is 32 bits (i.e., 4 octets). The information contained in each field of the message header is further explained below:



```
    0 byte          1 byte          2 byte          3 byte
|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Proto Type  |      0x01     |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| TLV 1 (Type)| TLV 1 (Length)  |         TLV 1 (Value)      ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| TLV 2 (Type)| TLV 2 (Length)  |         TLV 2 (Value)      ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4. TopoUpdate message format.



```
    0 byte          1 byte          2 byte          3 byte
|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Proto Type  |      0x02     |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       Payload (if any)                        ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 5. ReplyUpdate message format.

1. *Proto type*: Protocol type (8 bits). This field uses a specific hexadecimal number to denote the protocol type so that any switch that supports this protocol can easily identify SHP messages in the network.
2. *PDU type*: Packet Data Unit type (8 bits). This field specifies the type of message in the payload. For example, type 0×01 denotes a topoUpdate frame and type 0×02 indicates a replyUpdate frame.
3. *Message length*: Message size (16 bits). This field indicates the message end in the byte stream, starting from the first byte of the header.

5.1.1. *topoUpdate*

The topoUpdate message is used by the forwarding devices to announce that a network failure is affecting the connectivity established in the control plane. This message format was inspired by the use of Type-Length-Value (TLV) structures for the exchange of local neighbor information. TLV structures have been widely exploited by several existing standardized protocols, such as Link Layer Discovery Protocol (LLDP) [47], Intermediate System to Intermediate System (IS-IS) [48], Remote Authentication Dial-In User Service (RADIUS) [49], among others.

A TLV structure is a generic representation of an attribute that can be correctly parsed without requiring the parser to understand the attribute. Based on this, we utilized TLV as an efficient method for transmitting different kinds of topology data inside the message body. This encoding offers a reasonable balance between compactness and flexibility, which makes parsing faster and the data smaller. Moreover, using TLV for the data structure makes the proposed protocol extendable. Additionally, TLV elements can be placed in any order inside the message, which provides great flexibility in the design of the protocol.

While the TLV type and length fields occupy the first two octets of the TLV format, the value field may have a fixed or variable size. In addition, it may include different types of information, containing either binary or alpha-numeric data, which is specified using the associated subtype identifiers (e.g., port component, Media Access Control (MAC) or Internet Protocol (IP) address, interface name, locally assigned identifiers, etc.).

In Fig. 4, we provide the topoUpdate message format. Besides the header, this message carries two TLVs in the payload, namely, TLV node Identifier (ID) and TLV node port ID. The former is used to identify the node detecting the failure, while the latter specifies the involved port. Complementary TLVs can also be defined to enable protocol extensions.

This recovery message is first sent when a node detects a network failure in its parent port. This message is then forwarded through child ports to the downstream nodes along the compromised control branches. In addition, this message is also sent through the standby ports of each affected node as possible alternatives to recover the control path to an active SDN controller in the network. In this way, forwarding devices announce the network failure and simultaneously try to recover the control path toward an active SDN controller. By contrast, affected leaf and v-leaf nodes do not receive such a message, since no alternate path can be identified through them.

5.1.2. *replyUpdate*

When a forwarding device that has an active parent port receives a topoUpdate, rather than forwarding it, the node discards this message and responds with a replyUpdate. The data frame format of a replyUpdate is shown in Fig. 5.

Algorithm 1. topoUpdate message forwarding

Algorithm 1 topoUpdate message forwarding

```
1:  Node $v$ receives a $topoUpdate$ from node $u$ by port $p$
2:  if $p.state$ = $Parent$ or node $v$ does not have $Parent$ port then
3:      Set $Recovering$ state to port $p$ and $Child$ ports
4:      if same $topoUpdate$ had not been previously received then
5:          Forward $topoUpdate$ for all ports except $p$ or $Pruned$ ports
6:      else
7:          Discard $topoUpdate$ from node $u$                    ▹ to avoid propagation loops
8:      end if
9:  else
10:     Send $replyUpdate$ to node $u$ by port $p$                  ▹ without payload
11:     if same $topoUpdate$ had not been previously received then
12:         Send $replyUpdate$ for $Parent$ port toward the controller   ▹ with payload
13:     end if
14:     Discard $topoUpdate$ from node $u$                        ▹ to avoid propagation loops
15: end if
```

The replyUpdate message is critical in the process of recovering the broken control plane connectivity since its functionality is twofold. The presence of a payload in the replyUpdate message is optional, and its inclusion depends on the function performed by the particular instance of the message. If the payload is required, the information contained within it is either directly encapsulated by the node detecting the failure or taken from a previously received topoUpdate (i.e., TLV node ID and TLV node port ID).

First, this message is used to provide affected nodes with alternate control paths, enabling the reestablishment of the control connectivity in the hierarchical control tree. For this purpose, the replyUpdate message only carries its header information, so as to perform quick restoration and reduce the communication overhead. In particular, non-affected nodes, which become aware of the network failure after receiving a topoUpdate from a neighbor, send a short replyUpdate message to advertise themselves as possible points of recovery for the hierarchical control tree. In the same way, affected nodes also forward the first replyUpdate they receive across the disconnected branches, with the exception of their pruned ports. Similar to the previous message, affected leaf and v-leaf nodes do not receive this message because they are not able to provide a different route to reach the SDN controllers. In this way, forwarding additional messages to the nodes that cannot be used to recover the control tree topology is avoided.

The second task performed by the replyUpdate message is related to the notification of the network failure to the controllers. In this regard, the nodes with their active control path receive a topoUpdate from a neighbor, and also generate a second replyUpdate, which is sent through their parent port to the corresponding SDN controllers. The replyUpdate payload is reserved for this function, since in this case the information identifying the network failure (received in the topoUpdate) is included as part of the message. Therefore, the remainder of the nodes receiving this extended replyUpdate (i.e., those upstream nodes along the control path) also forward this message to the controller.

5.2. Mechanism operation

The forwarding devices initiate the proposed autonomic mechanism through the SHP. When a network device detects a port failure (i.e., when a neighbor's connectivity fails), the managed component executes specific actions depending on the state of its disrupted port (i.e., parent, child, pruned or standby).

Additionally, a new port state is defined in the SHP, called "recovering". This temporal port state identifies a forwarding port of an affected node that is connected to some disrupted network element (node or link). In particular, a disconnected node assigns the recovering state to those ports that are either part of the affected control branch or are receiving a topoUpdate from another affected neighbor.

Failures detected on standby, pruned or child ports are automatically reported to the SDN controllers with no change to the upstream control tree. To do this, the notification of the failure is sent through the established control branch to the corresponding SDN controller using an extended replyUpdate message. The failure is specified in this message, as are the respective identifiers of the node and port detecting the fault.

However, if the failure is detected through a parent port, the affected node must autonomously recover its control plane connectivity by making local decisions with no SDN controller intervention. First, the node informs its neighbors about the failure and forwards a topoUpdate message with its own node ID and involved node port ID through all the remaining ports except those that are pruned. Given their topological nature, leaf and v-leaf nodes cannot provide an alternative control path to the SDN controllers. Hence, unnecessary topoUpdate and replyUpdate messages are not forwarded to them in the control tree. This feature is critical to achieving a minimal communication overhead in the proposed SHP. The remainder of the process after receiving a topoUpdate message is described in Algorithm 1.

Algorithm 2. replyUpdate message forwarding

Algorithm 2 replyUpdate message forwarding

```
1:  Node $v$ receives a $replyUpdate$ from node $u$ by port $p$
2:  if node $v$ does not have $Parent$ port then
3:      Set $p.state = Parent$                          ▹ control plane connection of node $v$ is recovered
4:      Forward $replyUpdate$ for all the $Recovering$ ports
5:      Set $Standby$ state to all the $Recovering$ ports
6:      Send $ACK$ to node $u$ by port $p$                ▹ with updated topology information
7:  else if received $replyUpdate$ carries the failure identifiers then
8:      Forward $replyUpdate$ for the parent port toward the controller     ▹ with payload
9:  else
10:     Discard $replyUpdate$ from node $u$                ▹ to avoid propagation loops
11: end if
```

In essence, nodes receiving a topoUpdate message from their parent ports (or those that already have their parent ports disconnected), set the incoming port p, as well as all their child ports, to the recovering state (line 3). In this way, the nodes identify themselves as affected (i.e., in case the incoming port is the parent port) and mark the ports connected to neighbors that also require an alternate path to controllers. In addition, they propagate the received topoUpdate through all their ports, except the pruned ones, in order to notify their neighbors about the failure and identify a new path to the SDN controllers (line 5). The same sequence of actions is also performed by the node that has initially detected the failure from its parent port.

Nodes receiving a topoUpdate while their control paths are active (lines 9 to 5) discard this packet and answer it by sending a short replyUpdate to the affected neighbor. Next, these nodes notify the controller about the failure by sending an extended replyUpdate with the node and port identifiers received in the topoUpdate as a payload. It is important to note that, although a node may receive the same topoUpdate several times from different neighbors, the controller is informed only once about each particular failure.

As the announcement of the network failure is performed through the topoUpdate forwarding process, alternative control paths are advertised using the replyUpdate message. Algorithm 2 describes the steps after receiving a replyUpdate.

Once a disconnected node receives a replyUpdate, the neighbor sending this message becomes its point of recovery. This means that in order to provide a quick recovery strategy, each affected node will join with the neighbor from which it first receives the notification of an alternate control route (i.e., a short replyUpdate). Thus, the incoming port p is set to the parent state, indicating that node v has recovered its connection to the controller through this port (line 3). Then, the received replyUpdate is forwarded by all ports in the recovering state to notify other affected neighbors about this new possibility of reaching the controller (line 4). Afterward, these recovering ports are changed to the standby state (line 5). Furthermore, an acknowledge message (ACK) is sent by the affected node to its point of recovery in order to confirm this new association (line 6). Accordingly, the neighbor node, acting as a

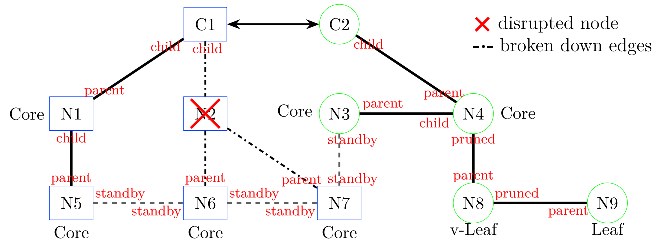


Fig. 6. Sample control tree topology with node failure.

point of recovery, changes the port status from standby to child.

When a node receives a replyUpdate with the network failure specified in the payload (i.e., for fault notification purposes), the incoming message is forwarded through the parent port to the controller (lines 7 and 8). Finally, although several replyUpdate messages can be sent to disconnected nodes from different neighbors, only the first is selected, meaning the replyUpdate messages received after the node is recovered are discarded (line 10).

Taking an SDN topology with two controllers and eight switches as an example, we can describe the basic operation of the SHP after a node failure occurs in the network. Specifically, in Fig. 6, we redraw the considered control tree topology, illustrating a sample disruption of the core node N2. In the explanation of this example, we are assuming that the closest active nodes to N6 and N7 are N5 and N3, respectively.

As shown in Fig. 6, when N2 fails, N6 and N7, which are connected to the disrupted node through their parent ports, lose their paths to the SDN controllers. Hence, both nodes send a topoUpdate message for all their active ports in order to announce the network failure and identify new control paths to the SDN controllers. Thus, two topoUpdate messages are propagated between neighbors, indicating the disrupted port of N6 in one and the disrupted port of N7 in the other.

When N6 receives the topoUpdate generated by N7, it changes the state of the incoming port of this message from the standby state to the recovering state. The same change is triggered in N7 after receiving the topoUpdate corresponding to N6. Once N5 and N3 become aware of the network disruption, they respond to the received topoUpdate packets, sending back two short replyUpdate messages to N6 and N7. In addition, each of these points of recovery (i.e., N5 and N3) notifies its controller about the network disruption using replyUpdate messages. In this instance, the replyUpdate messages are extended with the received TLVs specifying the failures.

After receiving the first replyUpdate, the two affected nodes assign the incoming ports to the parent state and forward these reply messages using their recovering ports. After doing this, ports in the temporary recovering state of both nodes are reset to the standby state. Additionally, these nodes reply to their respective points of recovery with an acknowledgment message. To avoid propagation loops, replyUpdate messages exchanged between the considered nodes (i.e., N6 and N7) and received after the control plane connection is recovered are discarded. The recovered control tree that results after the completion of this procedure is illustrated in Fig. 7.

To more clearly illustrate this process, Fig. 8 shows the message sequence for the proposed fault recovery mechanism when N6 detects that its parent port is nonfunctional. For the sake of simplicity, the exchange of recovery messages between N6 and N7 is not included in this figure.

As explained above, N6 sends one topoUpdate message containing its node ID and the node port ID of the port connected to N2, which has failed, to N5 and N7 simultaneously. This topoUpdate message is also forwarded from N7 to N3. Instead of forwarding the topoUpdate, active nodes that have their parent ports (i.e., N5 and N3) respond to this request by sending back a short replyUpdate message. This replyUpdate is sent through the path followed by the received topoUpdate message, creating a new way for N6 to reach the SDN controllers. At the same time, the received failure identifiers are sent by N5 to C1, using an extended replyUpdate.

Upon receiving the first replyUpdate sent by N5, N6 changes the state of the incoming port to the parent and automatically sends an acknowledgment message to N5. Afterward, any subsequent replyUpdate messages will be discarded (e.g., the one coming from N3 and forwarded by N7). The ACK message is used by N6 to announce to N5 that they are now joined in the recovered control tree topology. Hence, N5 should update its port status.

5.2.1. *Centralized optimization*

As connectivity is recovered from the broken state by only taking local actions, the network can be in a “good” but possibly degraded global state. After the control plane connectivity is recovered and the topology information of the SDN controllers is updated, the control paths can be centrally optimized. To achieve this, the autonomic managers, aware of the entire network view, may change the overlay control topology in order to improve performance (e.g., by finding the control paths with minimum delay).

In this regard, several optimization criteria may be considered in order to meet the requirements of the supported network applications and high-level objectives. In particular, due to the separation of network control from the forwarding devices, it is critical to establish control paths with minimum delay. To that end, minimizing propagation latency in control paths is fundamental to being able to respond to events in real time, and this has become one of the most significant design metrics for

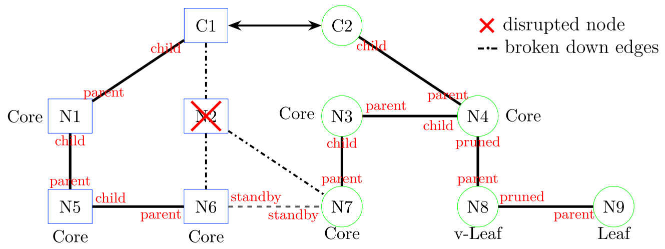


Fig. 7. Control topology recovered by SHP operation.

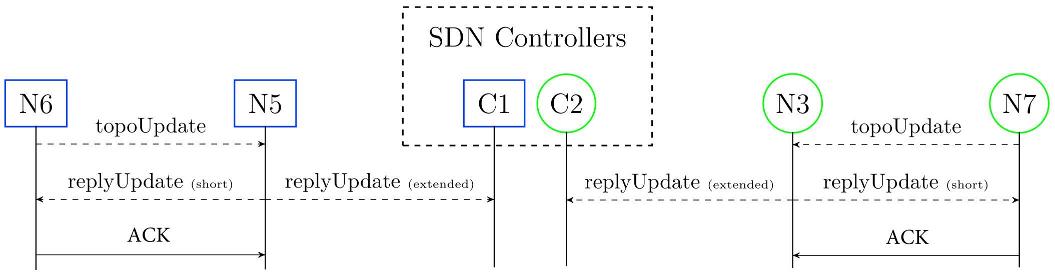


Fig. 8. Message flows for the proposed SHP.

the large-scale SDN.

The computation for minimizing propagation latency can be modeled using an optimal Integer Linear Programming (ILP). Designed to run as a network application on the SDN controller, this simple model computes a loop-free topology with optimal-delay control paths based on the network information previously collected by the controller. The goal is to identify the tree with the lowest path-delay between each node and the controller. Consequently, the topology information of the network could be sent to the controller using the shortest control paths (in terms of delay), allowing the topology data and statistic information of the forwarding plane to reach the controller with the shortest path-delay possible.

To describe the considered SDN, we model the network as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Each link (i,j) has its own associated non-negative delay $d_{i,j}$. We denote C as the controller location in a specific node of the network graph and U as the set of forwarding devices (i.e., $U = V \setminus C$). The goal of this model is to find the subset of control paths (P_C) that form a minimum-delay tree rooted in the SDN controller. To do this, the decision variable for the ILP model is defined as follows:

$p^u_{i,j}$: describes the selection of an edge (i,j) in the control path from a node $u \in U$ to the controller.

$$p^u_{i,j} = \begin{cases} 1, & \text{if edge } (i,j) \text{ is selected in the path,} \\ 0, & \text{otherwise.} \end{cases}$$

Using this notation, the objective function can be defined as follows:

$$\text{minimize} \quad \sum_{u \in U} \sum_{(i,j) \in E} p^u_{i,j} \cdot d_{i,j} \tag{1}$$

subject to:

$$\sum_{j \in N | (i,j) \in E} p^u_{i,j} - \sum_{j \in N | (j,i) \in E} p^u_{j,i} = \begin{cases} 1 & \text{if } i = u, \\ -1 & \text{if } i = C, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i \in N, \forall u \in U \tag{2}$$

Eq. (1) minimizes the delay in all control paths from each node to the controller. This objective function ensures an optimal delay spanning tree with the shortest control paths between each node and the SDN controller. Flow conservation constraints in Eq. (2) require that the control path of each node $u \in U$ is formed by the sequence of links at which $p^u_{i,j} = 1$. The overall control path delay is the summation of the corresponding selected link delays. Based on this simple formulation, we are able to find the optimal delay set of control paths from each node to the controller.

6. Evaluation and results discussion

In this section we first describe the simulation setup used to evaluate the proposed recovery mechanism. Then, we discuss the different tests conducted and results achieved in order to analyze the impact of the SHP on several network parameters.

6.1. Simulation environment

To implement the proposed solution, we used the discrete event simulator OMNeT++ [50] because of its suitability for studying realistic large-scale scenarios and because of the lack of suitable tools for researching the SDN from a layer 2 perspective [51,52].

For the conducted simulations, we worked with three network graphs representative of different scales from the available online dataset Survivable fixed telecommunication Network Design (SNDlib) [53]. Specifically, we selected Atlanta (15 nodes, 22 links), Sun (27 nodes, 51 links) and Pioro (40 nodes, 89 links). Other significant network parameters of these topologies are presented in Table 3.

Table 3
Network parameters of the topologies used in the simulations.

Topology	Nodes	Links	Average Degree	Diameter (*ms*)
Atlanta	15	22	2.93	3.1
Sun	27	51	3.78	3.7
Pioro	40	89	4.45	4.2

In order to evaluate the performance of our solution across varying connectivity degrees, we generated three family sets using these networks as seeds. Topologies that belong to each family set have been constructed as scale-free networks using a power-law node degree distribution with the same degree exponent as that of the original network. This was a result of using the static Barabási-Albert model [54] and maintaining the original number of nodes and links. Each family size was determined after restricting the margin of error of the indicated average values to less than 6% in each simulation instance. In particular, each topology set is composed of 500 generated networks. All simulation results include their respective 95% confidence intervals in the plots based on Student-t distribution. For SDN controller placements, we selected the most central nodes in each topology based on closeness centrality.

In our experimental simulations, we calculated the link propagation delays of the original topologies as the time needed for light to travel through the fiber. To do so, the distance between nodes was computed based on the locations provided in Ref. [53]. Then, for each network family, different link latencies were randomly generated, considering the mean and standard deviation values of the original topology used as the master. In addition to the propagation latency among devices, we also considered the packet processing time within each node. Specifically, the switch processing time was determined according to the sizes of messages as given in Ref. [55] for NetFPGA implementations.

6.1.1. BFD mechanism design

An essential element that is necessary to determine in our simulations in order to accurately assess the performance of the proposed fault recovery mechanism is the required latency to detect a link failure. Given that Ethernet is not designed with high requirements for failure detection, traditional techniques such as Loss of Signal (LoS) or layer 2 heartbeats cannot meet the 50 *ms* requirement of carrier-grade networks [56]. Therefore, in our simulations we have used a protocol-agnostic mechanism called BFD [35], for failure detection.

The goal of BFD is to provide low-overhead, short-duration detection of failures in links or paths between two end-point systems. This mechanism operates on top of any data protocol (e.g., network layer, link layer, tunnels, etc.) and is always executed in a unicast, point-to-point mode [35]. We have configured BFD sessions to detect link failures (between neighboring forwarding devices in the SDN) within the required 50 *ms*.

In Eq. (3), we derived the (worst-case) failure detection time T_{detec} using the BFD method.

$$T_{detec} = (M + 1) \cdot T_{interv} \tag{3}$$

As shown in Eq. (3), the failure detection time of the BFD strongly depends on the transmit interval T_{interv} (i.e., the periodicity of the control messages) and the detection time multiplier M. This parameter identifies when a session end-point is considered unreachable in terms of lost control packets. For the simulations, we utilized a multiplier of $M = 3$ to prevent small packet loss from triggering false positives.

Moreover, we focused on detecting link loss instead of path failures. Thus, only one BFD session was set per switch interface. This approach not only reduces detection time significantly but also decreases message complexity and overhead in the network. In Eq. (4), we derived the minimal transmit interval T_{min_interv} by implementing a BFD scheme that detects link losses instead of path failures, as previously explained. Link monitoring exhibits great improvement compared to per-path monitoring in terms of the failure detection time. This method is also adopted by Ref. [34].

$$T_{min\_interv} = 1.25 \cdot \beta \cdot T_{Round-Trip-Time(RTT)} \tag{4}$$

The transmit interval time is lower bounded by the RTT of a link in the network. On highly loaded links, this RTT measure can fluctuate greatly and might result in false positives [57]. Therefore, the retransmission interval of lost packets can be computed using $\beta \cdot T_{RTT}$, where β is the variation of the inter-arrival time. For our simulations, we selected a fixed and conservative value of $\beta = 2$, as identified in Ref. [58]. In addition, we validated the detection time measured in our experimental simulations using the analytical model presented above.

6.2. *Protocol performance*

In this subsection, we present the performance evaluation of the SHP solution for different key metrics and analyze the obtained results. Specifically, we assess the proposed SHP considering various metrics, such as recovery time, number of generated packets and percentage of nodes involved in the recovery process.

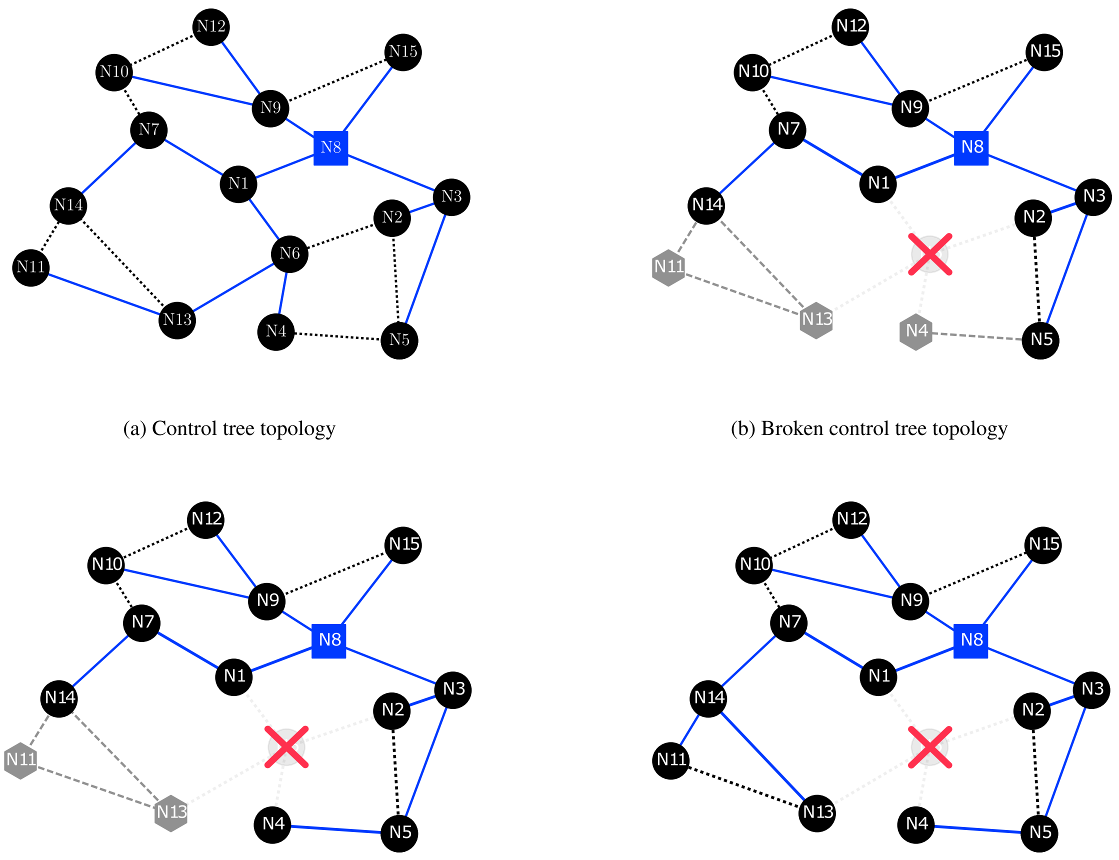
6.2.1. *SHP control tree recovering*

After a failure occurs in the network, the proposed mechanism attempts to recover the connectivity of the hierarchical control tree. To more clearly illustrate the operation of the SHP in the event of failures, we begin this evaluation section by presenting a basic recovering example using the Atlanta topology with a centralized controller (see Fig. 9).

For this evaluation, we placed the controller in the node denoted as N8, identified with a blue square. In Fig. 9(a), we first draw the hierarchical control tree originally created in this scenario, depicting the forwarding devices in the network as black circles. We use solid blue lines to represent the control paths established between the switches and the SDN controller in the tree. The remaining network links, not included in the control tree, are drawn using dotted black lines.

Next, in Fig. 9(b), we modify the previous graph to illustrate the occurrence of a node failure. Explicitly, the node denoted as N6 and its links are represented as partially transparent to identify this sample disruption. Additionally, nodes that lose their control connection due to the failure (i.e., nodes N4, N11 and N13), are depicted in a different shape (hexagon) and color (gray). Concerning the edges, the set of candidate links that can reestablish the affected control paths are identified using dashed gray lines. The exchange of topoUpdate and short replyUpdate messages performed by SHP occurs over these links.

Fig. 9(c) and (d) depict the recovering solution adopted for each of the two disconnected branches of the original control tree. In Fig. 9(c), we can see that the recovered control path of node N4 now goes through N5, which in this case is the only neighbor that sends a short replyUpdate to N4 with the notification of an alternate control route. In Fig. 9(d), however, we see that both nodes N11 and N13 receive the first replyUpdate message from the same point of recovery, namely N14. Therefore, two blue lines are drawn between these nodes to indicate the establishment of these new control paths. Meanwhile, the interfaces connecting the nodes N11 and N13 are now in the standby state.



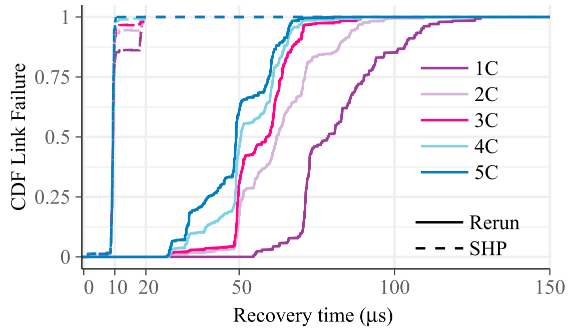
(a) Control tree topology

(b) Broken control tree topology

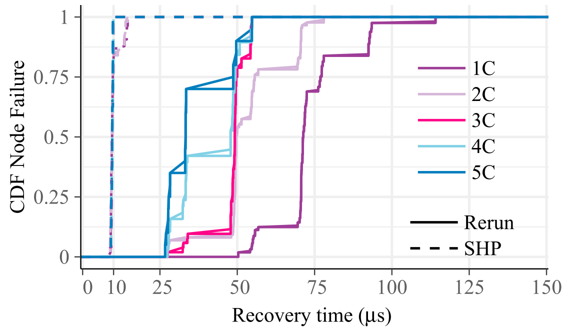
(c) Recovered control branch for N4

(d) Recovered control branches for N11 and N13

Fig. 9. Recovering of the control tree in Atlanta topology.



(a) Single link failure



(b) Single node failure

Fig. 10. Fault recovery time in Atlanta.

6.2.2. *Recovery time*

To assess the performance of the proposed SHP mechanism, we start by analyzing the recovery time. We have defined the SHP recovery time as the overall amount of time required to recover a disrupted control path. To be more detailed, this metric measures the period elapsed from the moment the failure occurs until the establishment of the new association between the disconnected node and one of its neighbors in the recovered control tree. Therefore, the recovery time is composed of the detection latency (obtained using the BFD strategy) and the time needed to complete the required SHP message exchange (i.e., from the sending of the first topoUpdate with the failure announcement to the reception of the acknowledgment message by the node acting as a point of recovery).

In Fig. 10, Fig. 11 and Fig. 12, we analyze single failure assumption for both links and nodes in the three selected topologies.

Note that a node failure corresponds to the occurrence of a multi-link failure. For this more complex scenario, the reported latencies reveal the entire period required to recover each of the individual link failures that comprise the network event.

In addition, in our simulations, we only consider the failure of links and nodes that do not affect the network connectivity, meaning the resulting graph remains strongly connected. In this way, we ensure that recovered control paths can always be established after the occurrence of the network failure.

To get a better sense of the achieved recovery time, we also include a Rerun approach in this analysis. This baseline approach refers to reapplying the topology discovery mechanism [45] after the SDN controllers are spontaneously notified of the network failure from the nodes that detect it. In this case, the recovery time is computed by considering the detection time, the time required to inform the controllers about the failure (using the corresponding shortest paths) and the discovery time.

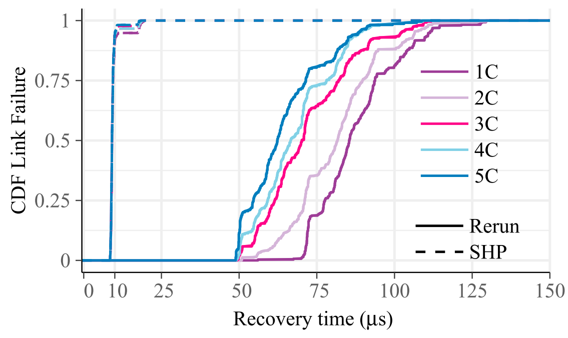
As expected, in all cases, the recovery mechanism outperforms the Rerun approach in terms of the required time to reestablish the connectivity of the hierarchical control tree. Specifically, for all the generated topologies, the fault recovery time is always below 20 μs for both considered cases (i.e., link and node failures). Therefore, the suitability of the recovery mechanism for application in carrier-grade networks, which requires less than the 50 *ms*, is confirmed. In addition, this behavior is not influenced by the increase of SDN controllers, validating the good scalability of this proposal.

6.2.3. *Recovery packets overhead*

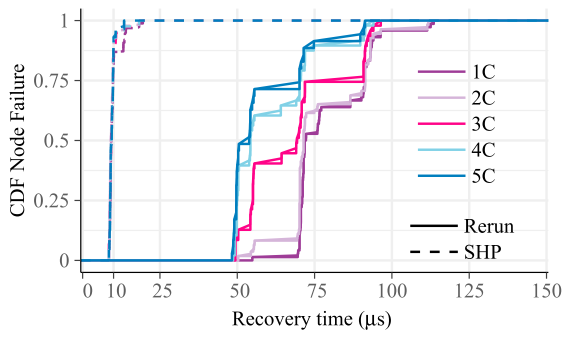
Next, we evaluate the impact of the proposed recovery strategy in terms of generated packets for various sizes of multi-link failures. In this analysis, we restrict the scope of the multiple failures to links connected to the same node because simultaneous wider-scope link failures are probably not realistic. In other words, we assume that simultaneous failures of multiple links are due to a failure of a node with a given connectivity degree. It should be noted that the failure of leaf and v-leaf nodes (nodes with a single way of reaching the controllers) are not included in this analysis since their control paths cannot be recovered.

Fig. 13 shows the average number of generated packets in comparison to the baseline Rerun strategy for the three considered topologies and varying the number of controllers. As previously mentioned, the degree of the failed node indicates the number of affected links. The number of packets reported in the plots represents the overall average of generated messages considering the failure of each node with a given connectivity degree for the 500 instances of a network.

From the results, it can be seen that under the SHP, smaller failures require the propagation of fewer messages, but this metric increases as the size of the failure (i.e., the number of affected links) increases. This result is expected given the generation of topoUpdate and replyUpdate messages defined by the SHP. In particular, under this strategy, the affected nodes try to recover their control paths as quickly as possible, and forward a topoUpdate message through each of their interfaces.



(a) Single link failure



(b) Single node failure

Fig. 11. Fault recovery time in Sun.

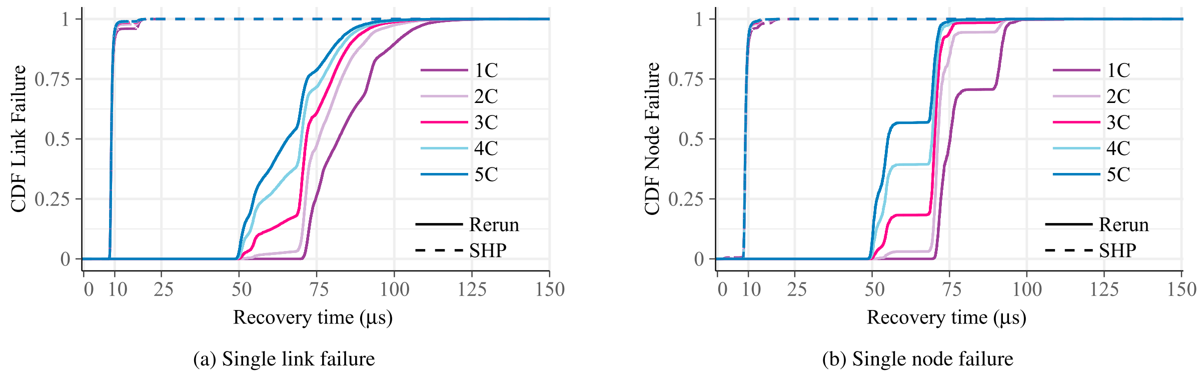


Fig. 12. Fault recovery time in Pioro.

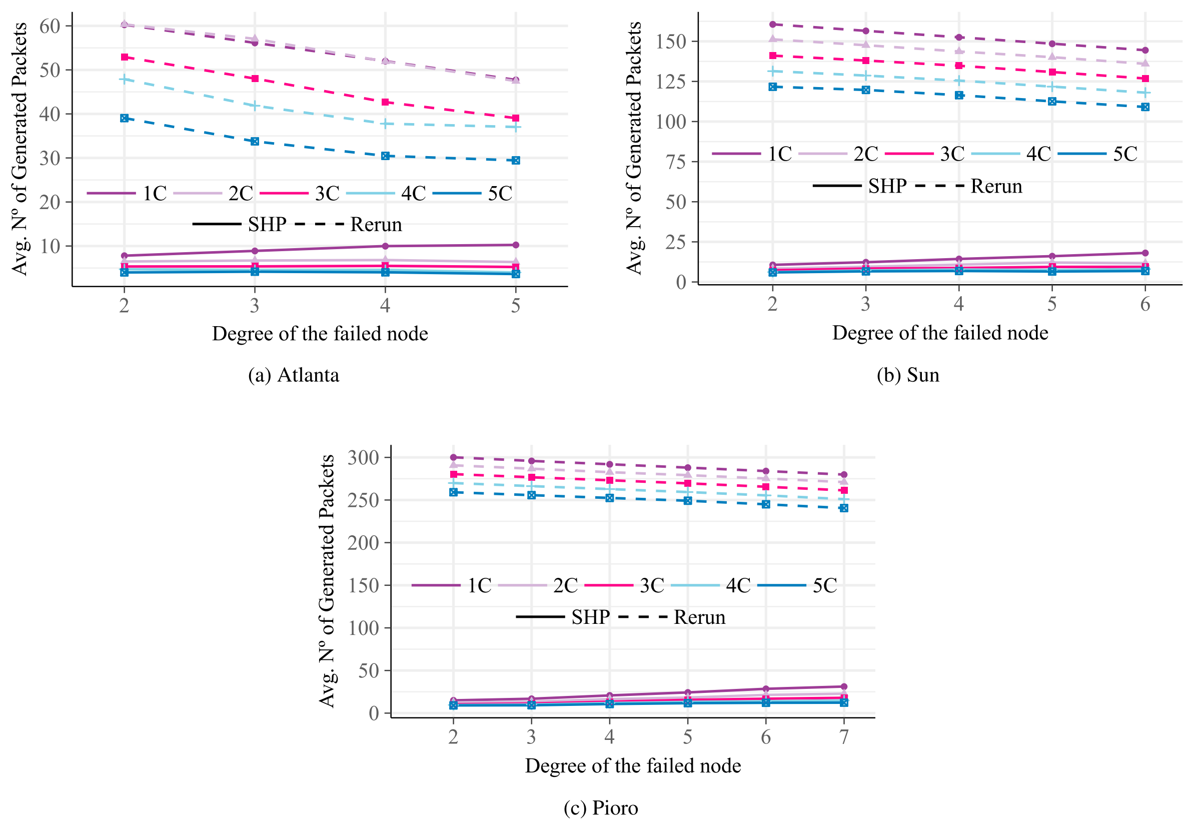


Fig. 13. Average number of generated packets for recovering multi-link failures.

Likewise, a short replyUpdate message is also received by every port of the affected node. Therefore, the degree of the failed node directly determines the resulted packet overhead. We can also observe from the figure that the increase in the number of generated packets corresponds to the network size. This result is due to the propagation of extended replyUpdate messages to notify the SDN controllers of the failure through the shortest path, the length of which (in terms of the number of hops) corresponds to the number of network nodes.

Furthermore, the same trend exists for different numbers of controllers. For small failures, the average number of messages is low and approximately the same for all controller values (around 5.67 in Atlanta, 7.69 in Sun and 11.32 in Pioro). However, a slight decrease in the number of generated packets can be observed as the number of controllers increases, and this difference becomes more noticeable when considering the failure of nodes at a higher degree. The reason for this is the reduction in the number of extended replyUpdate messages that are sent to announce the failure, as an increase in the number of controllers reduces the distance between them and the network nodes. In other words, when the number of controllers grows, fewer hops are likely needed to connect them with the neighbors of the affected node, which means that fewer replyUpdate messages are generated along these paths.

Inversely, under the Rerun strategy, the number of packets generated in the network corresponds with the average number of the node's neighbors. Therefore, given the reduction in the number of switches with higher connectivity degrees as a result of the considered node failure, the number of packets required for rediscovering the topology is decreased. We can see in this figure that, in all cases, the proposed recovery mechanism significantly outperforms the default Rerun strategy in terms of generated messages with percents of difference that are above 79%, 87% and 89%, respectively.

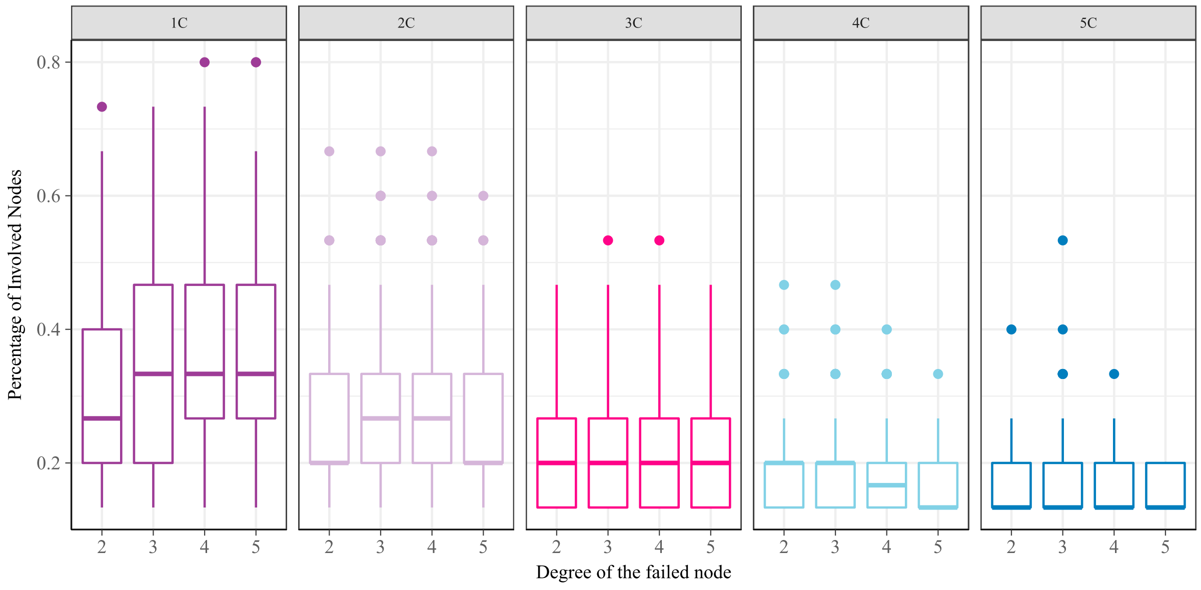


Fig. 14. Nodes involved in the SHP operation in Atlanta.

6.2.4. Number of involved nodes

In this last evaluation, we analyze the number of nodes involved in the operation of the proposed recovery mechanism. The term "involved nodes" takes into account the set of nodes performing different roles in the operation of SHP. Specifically, this set includes the disconnected nodes, their unaffected neighbors (i.e., those that can act as points of recovery since they have active parent ports) and the upstream nodes of those neighbors related to the SDN controllers. We use this metric to evaluate the impact of the SHP mechanism on the number of nodes with an additional workload as a result of the autonomous operation of this protocol.

Figs. 1415 and Fig. 16 show the number of involved nodes for different numbers of controllers and varying degrees of the affected node.

In this case, the Rerun strategy is not included in the plots because the operation of the topology discovery mechanism requires the implication of the entire network, increasing the workload of every switch. In contrast, the SHP reduces this impact by limiting the scope of the recovery functions to the neighborhood of nodes whose control plane connectivity has failed.

As shown, in the majority of cases for a given number of controllers, the number of switches involved in the recovery strategy increases while the degree of the affected node grows. This behavior is expected given that, in our approach, the neighbors of the disconnected node that still have active control paths are responsible for restoring the control plane connectivity.

Results also show that when the number of SDN controllers is increased, the number of involved switches decreases for a given failed node degree. As previously discussed, increasing the number of controllers reduces the length of branches in the control tree. As a result, a smaller number of nodes is required to send the failure notification to the network controllers.

In summary, in all the cases depicted in Figs. 14, Figs. 15 and 16, the

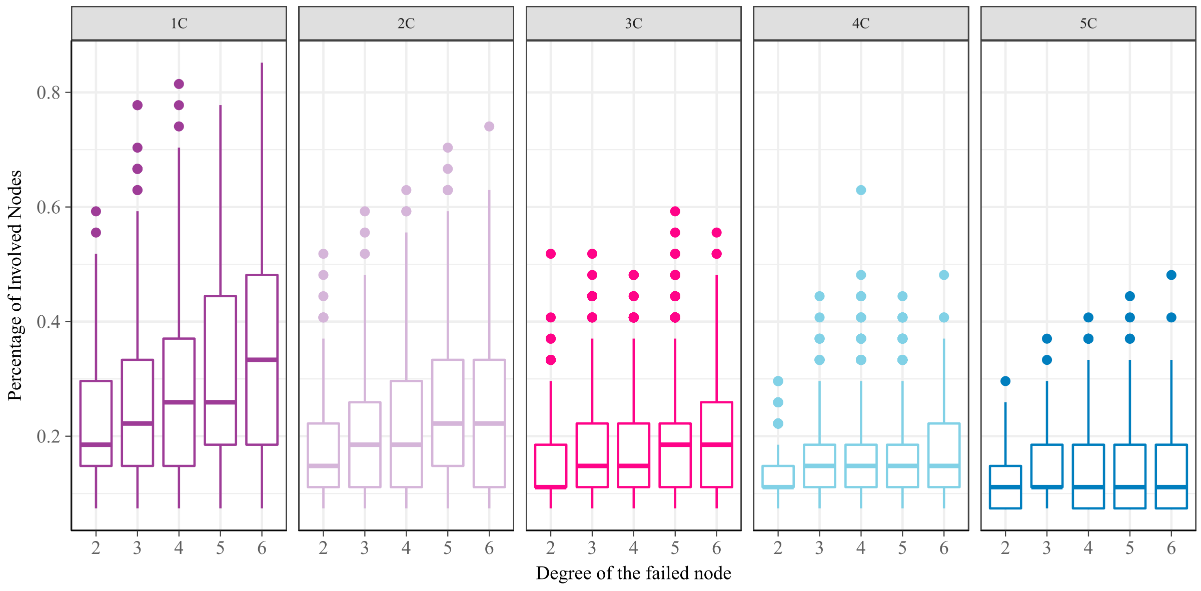


Fig. 15. Nodes involved in the SHP operation in Sun.

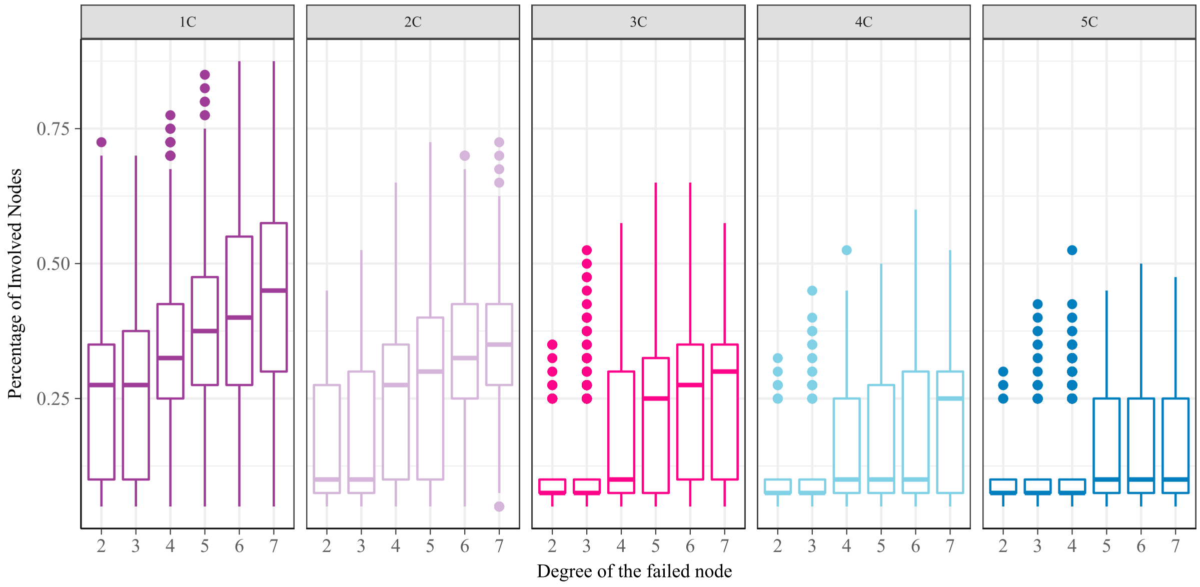


Fig. 16. Nodes involved in the SHP operation in Pioro.

average values of involved switches are always below 50% of the network nodes. This result reveals a significant merit of the SHP performance: it is able to achieve a fairly reasonable trade-off between reducing the recovery time with autonomic principles and keeping a minimal associated impact on the network devices in terms of the increased workload.

7. Conclusion

To address the concern of the resilience of the SDN, in this paper, we propose a self-healing mechanism that recovers the control plane connectivity in SDN-managed environments without overburdening the controller performance. The main idea underlying this proposal is to enable the real-time recovery of control paths in the face of failures without the intervention of a controller. To achieve this, we leverage the self-healing attribute of the ANM paradigm to guarantee the survivability of control connectivity as long as at least one SDN controller remains reachable within the network. The benefits of adopting the SHP are manifold. First, the mechanism uses the fewest possible number of messages (i.e., it has minimal communication overhead) and each message is small in size. Thus, it is easy to implement and yet efficient. Second, network devices can autonomously and stably recover the network from the “broken” states with no intervention of an SDN controller. In this way, not only is the workload of the SDN controllers minimized, the recovery time, packet loss probability and the memory requirements of forwarding devices are also reduced. In addition, once the connectivity is recovered throughout the control tree topology, the SDN controllers can optimize the recovered control plane by evaluating the requirements of the supported network applications. In addition, the results obtained in the experimental simulation reflect the time efficiency and scalability of the proposed solution across various key network metrics. Specifically, the recovery of the control connectivity was assured with the recovery time being below 20 μs for all the performed simulations. This result confirms the suitability of the recovery mechanism for application in carrier-grade networks, which require the recovery time to be less than 50 *ms*.

Conflict of interest

The authors declare that there is no conflict of interest regarding the publication of this article.

Acknowledgements

This work has been supported by the Ministerio de Economía y Competitividad of the Spanish Government under project TEC2016-76795-C6-1-R and AEI/FEDER, UE.

References

[1] S. Sezer, S. Scott-Hayward, P.K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, N. Rao, Are we ready for sdn? implementation challenges for software-defined networks, IEEE Commun. Mag. 51 (7) (2013) 36–43.
[2] J.H. Cox, J. Chung, S. Donovan, J. Ivey, R.J. Clark, G. Riley, H.L. Owen, Advancing software-defined networks: a survey, IEEE Access 5 (2017) 25487–25526, https://doi.org/10.1109/ACCESS.2017.2762291.
[3] L.H.J.W. Haibo Wang, Hongli Xu, X. Yang, Load-balancing routing in software defined networks with multiple controllers, Comput. Network. 141 (2018) 82–91.
[4] L. Ochoa-Aday, C. Cervelló-Pastor, A. Fernández-Fernández, Discovering the network topology: an efficient approach for sdn, Adv. Distr. Comput. Artif. Intell. J. 5 (2) (2016) 101–108.
[5] S. Sharma, D. Staessens, D. Colle, M. Pickavet, P. Demeester, Automatic bootstrapping of OpenFlow networks, in: Proc. Of the 19th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN), 2013, pp. 1–6. Brussels, Belgium.
[6] L. Schiff, S. Schmid, P. Kuznetsov, In-band synchronization for distributed sdn control planes, Comput. Commun. Rev. 46 (1) (2016) 37–43.
[7] Autonomic Computing, in: IBM's Perspective on the State of Information Technology, White Paper, IBM Press, 2001. URL, https://www.bibsonomy.org/bibtex/292d2eb8c354a1241e18416471572758c/neilernst.
[8] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, F. Zambonelli, A survey of autonomic communications, ACM Trans. Autonom. Adapt. Syst. 1 (2) (2006) 223–259.
[9] Z. Movahedi, M. Ayari, R. Langar, G. Pujolle, A survey of autonomic network architectures and evaluation criteria, IEEE Communications Surveys & Tutorials 14 (2) (2011) 464–490.
[10] N. Agoulmine, S. Balasubramaniam, D. Botvitch, J. Strassner, E. Lehtihet, W. Donnelly, Challenges for autonomic network management, in: Proc. Of the First IEEE International Workshop on Modelling Autonomic Communications Environments, MACE, Dublin, Ireland, 2006, pp. 1–20.
[11] M.C. Huebscher, J.A. McCann, A survey of autonomic computing—degrees, models, and applications, ACM Comput. Surv. 40 (3) (2008) 1–28.
[12] B. Jennings, S.V.D. Meer, S. Balasubramaniam, D. Botvich, M.O. Foghlu, W. Donnelly, J. Strassner, Towards autonomic management of communications networks, IEEE Commun. Mag. 45 (10) (2007) 112–121.
[13] R. Boutaba, J. Martin-Flatin, J. Hellerstein, R. Katz, G. Pavlou, C.-T. Lea, Recent advances in autonomic communications (Guest Editorial), IEEE J. Sel. Area. Commun. 28 (1) (2010) 1–3.
[14] W. Jiang, M. Strufe, H. Schotten, Autonomic network management for software-defined and virtualized 5G systems, in: Proc. Of the 23th European Wireless Conference, 2017, pp. 1–6. Dresden, Germany.
[15] N. Samaan, A. Karmouch, Towards autonomic network management: an analysis of current and future research directions, IEEE Communications Surveys & Tutorials 11 (3) (2009) 22–36, https://doi.org/10.1109/SURV.2009.090303.

[16] S. Kuklinski, P. Chemouil, Network management challenges in software-defined networks (Invited Paper), IEICE Trans. Commun. E97-B (1) (2014) 2–9.
[17] G. Poulios, K. Tsagkaris, P. Demestichas, A. Tall, Z. Altman, C. Destré, Autonomics and SDN for self-organizing networks, in: Proc. Of the 11th International Symposium on Wireless Communications Systems (ISWCS), 2014, pp. 830–835. Barcelona, Spain.
[18] S. Neti, H.A. Muller, Quality criteria and an analysis framework for self-healing systems, in: Proc. Of the International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'2007), 2007, pp. 1–10, https://doi.org/10.1109/SEAMS.2007.15. Minneapolis, MN, USA.
[19] H. Psaier, S. Dustdar, A survey on self-healing systems: approaches and systems, Computing 91 (1) (2011) 43–73, https://doi.org/10.1007/s00607-010-0107-y.
[20] D. Ghosh, R. Sharman, H. Raghav Rao, S. Upadhyaya, Self-healing systems – survey and synthesis, Decis. Support Syst. 42 (4) (2007) 2164–2185, https://doi.org/10.1016/j.dss.2006.06.011.
[21] I. Al-Oqily, S. Bani-Mohammad, B. Subaih, J.J. Alshaer, A survey for self-healing architectures and algorithms, in: Proc. Of the International Multi-Conference on Systems, Signals Devices (SSD'2012), Chemnitz, Germany, 2012, pp. 1–5, https://doi.org/10.1109/SSD.2012.6198057.
[22] J.-P. Vasseur, M. Pickavet, P. Demeester, Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
[23] P.C.d.R. Fonseca, E.S. Mota, A survey on fault management in software-defined networks, IEEE Communications Surveys & Tutorials 19 (4) (2017) 2284–2321, https://doi.org/10.1109/COMST.2017.2719862.
[24] Open Networking Foundation, OpenFlow Switch Specification v1.4.0, Technical Specification, Oct. 2013. URL, https://www.opennetworking.org/.
[25] N.M. Sahri, K. Okamura, Fast failover mechanism for software defined networking: OpenFlow based, in: Proc. Of the Ninth International Conference on Future Internet Technologies (CFI'2014), Tokyo, Japan, 2014, https://doi.org/10.1145/2619287.2619303, 16:1–16:2.
[26] Y. Lin, H. Teng, C. Hsu, C. Liao, Y. Lai, Fast failover and switchover for link failures and congestion in software defined networks, in: Proc. Of the IEEE International Conference on Communications (ICC'2016), Kuala Lumpur, Malaysia, 2016, pp. 1–6, https://doi.org/10.1109/ICC.2016.7510886.
[27] S. Sharma, D. Staessens, D. Colle, M. Pickavet, P. Demeester, In-band control, queuing, and failure recovery functionalities for Openflow, IEEE Network 30 (1) (2016) 106–112, https://doi.org/10.1109/MNET.2016.7389839.
[28] E.G. Pereira, R. Pereira, A. Taleb-Bendiab, Performance evaluation for self-healing distributed services and fault detection mechanisms, J. Comput. Syst. Sci. 72 (7) (2006) 1172–1182, https://doi.org/10.1016/j.jcss.2005.12.008.
[29] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, B.B. Kang, Rosemary: a robust, secure, and high-performance network operating system, in: Proc. Of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Scottsdale, Arizona, USA, 2014, pp. 78–89, https://doi.org/10.1145/2660267.2660353.
[30] A. Capone, C. Cascone, A.Q.T. Nguyen, B. Sansò, Detour planning for fast and reliable failure recovery in SDN with OpenState, in: Proc. Of the 11th International Conference on the Design of Reliable Communication Networks (DRCN), 2015, pp. 25–32. Kansas City, MO, USA.
[31] C. Cascone, L. Pollini, D. Sanvito, A. Capone, Traffic management applications for stateful SDN data plane, in: Proc. Of the Fourth European Workshop on Software-Defined Networks, 2015, pp. 85–90, https://doi.org/10.1109/EWSDN.2015.66. Bilbao, Spain.
[32] P. Thorat, S. Jeon, H. Choo, Enhanced local detouring mechanisms for rapid and lightweight failure recovery in openflow networks, Comput. Commun. 108 (2017) 78–93, https://doi.org/10.1016/j.comcom.2017.04.005.
[33] P. Thorat, S.M. Raza, D.S. Kim, H. Choo, Rapid recovery from link failures in software-defined networks, J. Commun. Netw. 19 (6) (2017) 648–665, https://doi.org/10.1109/JCN.2017.000105.
[34] N.L. M.v. Adrichem, B.J.v. Asten, F.A. Kuipers, Fast recovery in software-defined networks, in: Proc. Of the 3rd European Workshop on, Software-Defined Networks, Budapest, Hungary, 2014, pp. 61–66.
[35] D. Katz, D. Ward, in: Bidirectional Forwarding Detection (BFD), RFC 5880, RFC Editor, June 2010. URL, http://www.ietf.org/rfc/rfc5880.txt.
[36] B. Raeisi, A. Giorgetti, Software-based fast failure recovery in load balanced SDN-based datacenter networks, in: Proc. Of the 6th International Conference on Information Communication and Management (ICICM), 2016, pp. 95–99, https://doi.org/10.1109/INFOCOMAN.2016.7784222. Hatfield, UK.
[37] P. Thorat, S.M. Raza, D.T. Nguyen, G. Im, H. Choo, D.S. Kim, Optimized self-healing framework for software defined networks, in: Proc. Of the 9th International Conference on Ubiquitous Information Management and Communication (IMCOM), Bali, Indonesia, 2015, https://doi.org/10.1145/2701126.2701235, 7:1–7:6.
[38] J.M. Sánchez, I.G.B. Yahia, N. Crespi, T.M. Rasheed, D. Siracusa, Softwarized 5g Networks Resiliency with Self-Healing, CoRR Abs/1507.02951, URL, http://arxiv.org/abs/1507.02951.
[39] J. M. Sánchez, I. G. B. Yahia, N. Crespi, POSTER: Self-healing mechanisms for software-defined networks, CoRR abs/1507.02952. URL http://arxiv.org/abs/1507.02952
[40] J.M. Sánchez, I.G.B. Yahia, N. Crespi, THESARD: on the road to resilience in software-defined networking through self-diagnosis, in: Proc. Of the IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 351–352, https://doi.org/10.1109/NETSOFT.2016.7502406. Seoul, South Korea.
[41] L. Ochoa-Aday, C. Cervelló-Pastor, A. Fernández-Fernández, Self-healing topology discovery protocol for software defined networks, IEEE Commun. Lett. 22 (5) (2018) 1070–1073.
[42] ETSI, in: An Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management, ETSI GS AFI 002 V1.1.1, Apr. 2013. URL, https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf.
[43] S. Khan, A. Gani, A.A. Wahab, M. Guizani, M.K. Khan, Topology discovery in software defined networks: threats, taxonomy, and state-of-the-art, IEEE Communications Surveys & Tutorials 19 (1) (2017) 303–324, https://doi.org/10.1109/COMST.2016.2597193.
[44] X. Yu, H. Xu, D. Yao, H. Wang, L. Huang, CountMax: a lightweight and cooperative sketch measurement for software-defined networks, IEEE/ACM Trans. Netw. 26 (6) (2018) 2774–2786.
[45] L. Ochoa-Aday, C. Cervelló-Pastor, A. Fernández-Fernández, eTDP: enhanced topology discovery protocol for software-defined networks, IEEE Access 7 (2019) 23471–23487.
[46] A. Fernández-Fernández, C. Cervelló-Pastor, L. Ochoa-Aday, Improved energy-aware routing algorithm in software-defined networks, in: Proc. Of the 41st IEEE Conference on Local Computer Networks, LCN, 2016, pp. 196–199.
[47] IEEE Standard for Local and Metropolitan Area Networks– Station and Media Access Control Connectivity Discovery, Mar. 2016, https://doi.org/10.1109/IEEESTD.2016.7433915.
[48] R. Callon, in: Use of OSI IS-IS for Routing in TCP/IP and Dual Environments, RFC 1195, RFC Editor, Dec. 1990. URL, http://www.rfc-editor.org/rfc/rfc1195.txt.
[49] A. DeKok, A. Lior, in: Remote Authentication Dial in User Service (RADIUS) Protocol Extensions, RFC 6929, RFC Editor, Apr. 2013. URL, http://www.rfc-editor.org/rfc/rfc6929.txt.
[50] OMNeT++ – Discrete Event Simulator (version 5.4.1), (accessed on October 19, 2018). URL https://www.omnetpp.org/
[51] G. Anggono, T. Moors, A flow-level extension to OMNeT++ for long simulations of large networks, IEEE Commun. Lett. 21 (3) (2017) 496–499, https://doi.org/10.1109/LCOMM.2016.2628356.
[52] A.W. Malik, K. Bilal, S.U. Malik, Z. Anwar, K. Aziz, D. Kliazovich, N. Ghani, S.U. Khan, R. Buyya, CloudNetSim++: a GUI based framework for modeling and simulation of data centers in OMNeT++, IEEE Transactions on Services Computing 10 (4) (2017) 506–519, https://doi.org/10.1109/TSC.2015.2496164.
[53] S. Orlowski, M. Pióro, A. Tomaszewski, R. Wessäly, SNDlib 1.0-survivable network design library, Networks 55 (3) (2010) 276–286, https://doi.org/10.1002/net.20371.
[54] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512, https://doi.org/10.1126/science.286.5439.509.
[55] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, P. Tran-Gia, Modeling and performance evaluation of an OpenFlow architecture, in: Proc. Of the 23rd International Teletraffic Congress (ITC), 2011, pp. 1–7. San Francisco, CA, USA.
[56] S. Sharma, D. Staessens, D. Colle, M. Pickavet, P. Demeester, OpenFlow: meeting carrier-grade recovery requirements, Comput. Commun. 36 (6) (2013) 656–665, https://doi.org/10.1016/j.comcom.2012.09.011.
[57] V. Jacobson, Congestion avoidance and control, Comput. Commun. Rev. 18 (4) (1988) 314–329, https://doi.org/10.1145/52325.52356.
[58] D. Clark, in: Window and Acknowledgement Strategy in TCP, RFC 813, RFC Editor, July 1982. URL, http://www.rfc-editor.org/rfc/rfc813.txt.