Enabling Delegation of Control Plane Functionalities for Time Sensitive Networks

N. SAMBO¹, S. FICHERA¹, A. SGAMBELLURI¹, G. FIOCCOLA², P. CASTOLDI¹, K. KATSALIS²

¹Scuola Superiore Sant'Anna, Via G. Moruzzi, 1, 56124, Pisa, Italy (nicola.sambo@sssup.it).
²Huawei Technologies Duesseldorf Gmbh, Munich, Germany.

Corresponding author: Nicola Sambo (e-mail: nicola.sambo@sssup.it).

The authors acknowledge Muhammad Usman who contributed to the data plane deployment.

ABSTRACT This paper proposes a new paradigm for control plane in Time Sensitive Networks (TSN). An SDN controller proactively instructs network elements on the reconfigurations to perform locally if some specific events occur (e.g., failures, performance degradations). Instructions are given in the form of Finite State Machines (FSMs), which store information related to the actions that each network element should execute to react against a specific event. Thus, if such events occur, the SDN controller is by-passed reducing reaction (e.g., recovery) time. Such an approach is here implemented for recovery upon failures in TSN. Experiments of failure recovery are carried out and measurements are presented comparing the FSM-based solution with a fully-centralized reactive restoration. Moreover, the proposed approach is compared through simulations against Frame Replication and Elimination for Reliability. Results will show how proactive FSM manipulation can strongly reduce recovery time in SDN-based TSN networks without overloading the network with frame replicas.

INDEX TERMS TSN, Time Sensitive Networking, IEEE 802.1, SDN, NETCONF, YANG, recovery, restoration, FRER.

I. INTRODUCTION

Emerging services supported by a number of challenging applications like industrial automation, or autonomous driving, require efficient control and management of network resources to assure requirements not only by means of throughput but also by means of delay and jitter. Time Sensitive Networks (TSNs) like the ones defined by IEEE TSN Working Group [1], are able to satisfy stringent requirements by means of reliability, delay, and jitter. However, still there is no consensus within the industry on which control plane (fully centralized or fully distributed) is ideal for TSN. Regarding a fully centralized scenario, following the Software Defined Networking (SDN) paradigm [2], a number of benefits like rapid service deployment and global network optimization can be achieved. However, several issues are still open, e.g. concerning scalability and dynamic restoration [3], especially in TSN. For example, for large networks, in the presence of failures, fast network recovery is not easy to achieve.

The reason is that a huge number of restoration requests may need to be processed by the SDN controller, thus greatly affecting recovery time [4]. In order to avoid such problems and thus to enhance network reliability in TSN, more robust mechanisms can be also adopted, as Frame Replication and Elimination for Reliability (FRER) [5]. FRER is based on redundant transmissions along alternative routes of each flow. Although using FRER indisputably increases reliability, the drawback is the increased network load due to replicas and an higher scheduling complexity especially in the presence of time critical flows. Moreover, in general, in SDN networks, several recovery mechanisms – like Fast Failover [6] – have been implemented improving responsiveness in OpenFlow-based networks [4], [6], [7]. However, there is a lack of methods which are compliant with the NETCONF protocol [8] and YANG data modelling [9], which are considered and already supported in TSN networks [10]–[12], especially to offer open and vendor-independent network configur-
Recently, within the Internet Engineering Task Force (IETF), a NETCONF/YANG method [8], [9] has been proposed to describe and manage events, operations, and states related to a network [13]. Such method is based on Finite State Machines (FSMs), which store information related to events, network states, operations and network reconfiguration. FSM can be installed by an SDN controller in a device agent which is responsible to autonomously perform operations associated to a specific network state. According to [13], the dynamic FSM loading mechanism can be used in a variety of use cases and applications, including: i) recovery and transmission adaptation in optical networks [14], [15]; ii) network telemetry to define and embed custom data probes into data plane devices; iii) monitoring optimization of packet loss and delay for real time systems.

In this work, in a context of an SDN scenario, we propose to transfer – through the NETCONF protocol [8] and FSM – control logics into the agents of TSN devices. Thus, the SDN controller proactively instructs the agents deployed at each network element, while the network is properly operating, on the actions to perform in case of specific events. This way, devices’ agents are able to autonomously take the proper actions without the SDN controller intervention, thus reducing control plane closed-loop delays due to signaling and also due to processing at the SDN controller. We showcase the FSM technique in TSN networks for a reliability use case, proposing a recovery mechanism named Delegated Restoration for TSN (DRTSN).

The contributions of this paper are the following. DRTSN based on FSM is presented for recovery in TSN. The implementation of agent modules and of the data plane is detailed. An experimental testbed is set up and the proposed method analyzed through measurements (e.g., recovery delay) against a fully centralized reactive restoration. The analysis of time contributions to perform agent functions is shown. After the observation of measurements, an optimization of the whole procedure is provided showing a further reduction in recovery delay achieved with DRTSN. Then, DRTSN is compared with FRER by means of simulations. Measurements and simulations show that DRTSN permits to achieve faster recovery time than the reactive restoration, without overloading the network as FRER.

The paper is organized as follows. We present background information and related work in section II. In section III, we present the system architecture and the adopted FSM YANG model. In section IV, DRTSN for recovery in TSN is detailed. In section V we present the implementation in Linux. In section VI we evaluate the performance of the proposed solution. We conclude our study and present future research directions in section VII.

II. BACKGROUND INFORMATION AND RELATED WORK

**TSN Data Plane:** Techniques used to provide delay guarantees are Scheduled Traffic (IEEE 802.1Qbv [16]), Frame Preemption (IEEE 802.3br [17], IEEE 802.1Qbu [18]), Asynchronous Traffic Shaping (802.1Qcr [19]) and Cyclic Queueing and forwarding (802.1Qch [20]). These standards define how frames belonging to a particular traffic class or having a particular priority are handled by TSN-enabled bridges. We focus on IEEE 802.1Qbv [16] which introduces a transmission gate operation for each traffic class queue. The transmission gates are in *open* or *close* state and controlled by a Gate Control List (GCL). For each output port a GCL consists of multiple schedule entries. For the open gates, selected traffic is allowed to pass through to the transmission selection block, which provides access to the medium. Frame Preemption, allows the ongoing transmission of a lower priority frame to be preempted by a higher priority frame (express traffic) and thus ensures lower latency for high priority frames. Express frames can preempt preemptable frames by either interrupting the frame transmission or by preventing the start of a pMAC frame transmission. The queuing model is different in case that ATS scheduling is applied (802.1Qcr specification [19]).

Regarding 802.1Qbv scheduling, we refer interested readers to [21]. In [22], the TSN 802.1Qbv scheduling problem is addressed by exploiting techniques, such as Satisfiability Modulo Theory (SMT) and Optimization Modulo Theory (OMT). Delay analysis for AVB traffic in 802.1Qbv is presented in [24], [25]. More recently, Window-Based Schedule Synthesis [26] has been proposed for industrial IEEE 802.1Qbv TSN Networks with unscheduled end-systems. In [27] the problem of finding the routes for AVB flows over TSN-based networks is addressed. The authors use a K-shortest path heuristic to reduce the search space of routes and a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic for optimizing the routing. Authors in [28] explore how the routing of time-triggered flows affects their schedulability and also propose ILP-based algorithms for constructive deterministic routes. Various ILP formulations for the combined routing and scheduling time-triggered traffic problem, while following the SDN-based paradigm are presented in [29]. Authors in [30] propose a joint routing and scheduling approach for both TT and AVB traffic. An ILP scheduling and routing formulation to improve the TT traffic schedulability is proposed in [31]. A List Scheduling-based heuristic and scheduling and routing is proposed in [32]. The work in [33] has shown how to reconfigure the GCLs at runtime, e.g., as a reaction to network changes, which could also be due to failures.

**TSN Control plane:** three models have been proposed in the 802.1Qcc amendment. In the fully centralized case, the flow requirements are conveyed from a Centralized User Configurator (CUC) to a Centralised Network Configurator (CNC), using a User Network Information (UNI) described in 802.1Qdd. The CNC is responsible for the configuration and control of the TSN switch fabric, while the CUC is responsible for the end-points (Talkers/Listeners). In the 802.1Qdl amendment, the fully distributed case is investigated, while in the Centralized Network-Distributed User Model, the UNI exists between the endpoints and the access...
Fast Failover has been proposed [6]. It leverages the concept of Groups, where each group is composed of a series of flows, all treated in the same way. Fast Failover is a proactive approach that establishes switching rules before failure with the objective of speeding up recovery time. A similar approach has been proposed in [4], also managing congested links. In [7], a segment-based recovery has been proposed for OpenFlow. Basically, with these approaches, a node detecting the failure, switches the traffic to another port, i.e. the one associated to the backup route. In general, such methods may suffer from the fact that, if the node detecting the failure does not have any other ports (e.g., as in a ring topology) to switch traffic, central controller intervention may be required, thus resulting in a reactive method. In general, previous approaches in SDN networks have been mainly implemented with the OpenFlow protocol.

Moreover, hybrid SDN has been investigated. In [36], hybrid SDN is referred to networks with sparse SDN switches in a legacy network or to a network composed of switches having both SDN switching and legacy switching functionalities (thus, based on both SDN and distributed protocols). As an example, in [37], a recovery mechanism is proposed in an environment where legacy network devices and SDN switches co-exist. In [38], a centralized controller is used for long-term optimization relying on OpenFlow to configure arbitrary forwarding paths during normal operations, while distributed protocols (e.g., Interior Gateway Protocol) can be used for short-term reaction to failures. Hybrid SDN may provide more robustness against failures than classical SDN. In our paper, the proposed solution does not exploit legacy-network distributed protocols, while it operates with the only NETCONF/YANG in both normal and failure conditions.

A different approach in order to enhance reliability is to rely on redundant transmissions. IEEE 802.1Qbd amendment the fully distributed case is investigated where the resource allocation and registration is made in a distributed fashion, while the endpoints only interact with the access TSN bridge over the UNI.

Failure recovery: Recovery in SDN networks is classified in two main categories: proactive and reactive [4]. With the proactive methods, one or more alternative paths are computed before the failure, while the reactive methods require that switches send a restoration request to the central controller. Proactive approaches permit to reduce recovery time with respect to reactive approaches. Reliable Control and Data Planes for SDN have been investigated in [34]. A method of handling data packets through a conditional state transition table for implementing at least one finite state machine is proposed by [35]. In OpenFlow environments, Fast Failover has been proposed [6]. It leverages the concept of Groups, where each group is composed of a series of flows, all treated in the same way. Fast Failover is a proactive approach that establishes switching rules before failure with the objective of speeding up recovery time. A similar approach has been proposed in [4], also managing congested links. In [7], a segment-based recovery has been proposed for OpenFlow. Basically, with these approaches, a node detecting the failure, switches the traffic to another port, i.e. the one associated to the backup route. In general, such methods may suffer from the fact that, if the node detecting the failure does not have any other ports (e.g., as in a ring topology) to switch traffic, central controller intervention may be required, thus resulting in a reactive method. In general, previous approaches in SDN networks have been mainly implemented with the OpenFlow protocol.

Moreover, hybrid SDN has been investigated. In [36], hybrid SDN is referred to networks with sparse SDN switches in a legacy network or to a network composed of switches having both SDN switching and legacy switching functionalities (thus, based on both SDN and distributed protocols). As an example, in [37], a recovery mechanism is proposed in an environment where legacy network devices and SDN switches co-exist. In [38], a centralized controller is used for long-term optimization relying on OpenFlow to configure arbitrary forwarding paths during normal operations, while distributed protocols (e.g., Interior Gateway Protocol) can be used for short-term reaction to failures. Hybrid SDN may provide more robustness against failures than classical SDN. In our paper, the proposed solution does not exploit legacy-network distributed protocols, while it operates with the only NETCONF/YANG in both normal and failure conditions.

A different approach in order to enhance reliability is to rely on redundant transmissions. IEEE 802.1Qbd amendment the fully distributed case is investigated where the resource allocation and registration is made in a distributed fashion, while the endpoints only interact with the access TSN bridge over the UNI.

The DRTSN method based on FSM proposed in this paper falls within the umbrella of proactive schemes. It does not require any central controller intervention upon failure neither packets/frames replica, and it is specifically designed/implemented for the NETCONF protocol and YANG data modelling language. User-defined FSM enforcement can be also applied using P4 language and compilers [39]. We plan to incorporate the concept of FSM with P4 over a stateless data plane in our future work.

III. TECHNICAL APPROACH

We assume a network composed of multiple forwarding devices with TSN aware bridging capabilities and several talker/listener pairs, generating and receiving traffic respectively. The solution we will present can be applied in both pure L2 or L3 operations as long as there is TSN support on the forwarding plane. During normal operation, the SDN controller proactively configures the specific FSM in the local Agent deployed at each device. A FSM includes information related to the set of actions to be locally taken to re-act against an event. For example, a node detecting a failure triggers a state transition into its FSM, from a “Stable” state to “Failure” state. Then, such new state implies a set of specific actions to be taken in order to react against the event. FSM is generic and can model any system characteristic or event. This paper will be focused on the recovery use case in TSN. Depending on the state, we assume that the set of actions can be: a) TSN related configuration, b) L2/L3 forwarding rules.

A. SYSTEM ARCHITECTURE

For each forwarding device, the system architecture we consider is depicted in Fig.1. The proposed solution is thought for the fully centralized control plane architecture presented in IEEE 802.1Qcc [40]. Please note that a fully centralized architecture may be required since, as reported in the standard, many TSN use cases require significant user configuration in the end stations (talkers and listeners), such as in many automotive and industrial control applications. In such use cases, treatment of the reversion of Group membership is required. This is done by the use of an OpenFlow controller, which stores the state of all Groups, where each group is composed of a series of flows, all treated in the same way. Fast Failover is a proactive approach that establishes switching rules before failure with the objective of speeding up recovery time. A similar approach has been proposed in [4], also managing congested links. In [7], a segment-based recovery has been proposed for OpenFlow. Basically, with these approaches, a node detecting the failure, switches the traffic to another port, i.e. the one associated to the backup route. In general, such methods may suffer from the fact that, if the node detecting the failure does not have any other ports (e.g., as in a ring topology) to switch traffic, central controller intervention may be required, thus resulting in a reactive method. In general, previous approaches in SDN networks have been mainly implemented with the OpenFlow protocol.

Moreover, hybrid SDN has been investigated. In [36], hybrid SDN is referred to networks with sparse SDN switches in a legacy network or to a network composed of switches having both SDN switching and legacy switching functionalities (thus, based on both SDN and distributed protocols). As an example, in [37], a recovery mechanism is proposed in an environment where legacy network devices and SDN switches co-exist. In [38], a centralized controller is used for long-term optimization relying on OpenFlow to configure arbitrary forwarding paths during normal operations, while distributed protocols (e.g., Interior Gateway Protocol) can be used for short-term reaction to failures. Hybrid SDN may provide more robustness against failures than classical SDN. In our paper, the proposed solution does not exploit legacy-network distributed protocols, while it operates with the only NETCONF/YANG in both normal and failure conditions.

A different approach in order to enhance reliability is to rely on redundant transmissions. IEEE 802.1Qbd amendment the fully distributed case is investigated where the resource allocation and registration is made in a distributed fashion, while the endpoints only interact with the access TSN bridge over the UNI.

Failure recovery: Recovery in SDN networks is classified in two main categories: proactive and reactive [4]. With the proactive methods, one or more alternative paths are computed before the failure, while the reactive methods require that switches send a restoration request to the central controller. Proactive approaches permit to reduce recovery time with respect to reactive approaches. Reliable Control and Data Planes for SDN have been investigated in [34]. A method of handling data packets through a conditional state transition table for implementing at least one finite state machine is proposed by [35]. In OpenFlow environments, Fast Failover has been proposed [6]. It leverages the concept of Groups, where each group is composed of a series of flows, all treated in the same way. Fast Failover is a proactive approach that establishes switching rules before failure with the objective of speeding up recovery time. A similar approach has been proposed in [4], also managing congested links. In [7], a segment-based recovery has been proposed for OpenFlow. Basically, with these approaches, a node detecting the failure, switches the traffic to another port, i.e. the one associated to the backup route. In general, such methods may suffer from the fact that, if the node detecting the failure does not have any other ports (e.g., as in a ring topology) to switch traffic, central controller intervention may be required, thus resulting in a reactive method. In general, previous approaches in SDN networks have been mainly implemented with the OpenFlow protocol.

Moreover, hybrid SDN has been investigated. In [36], hybrid SDN is referred to networks with sparse SDN switches in a legacy network or to a network composed of switches having both SDN switching and legacy switching functionalities (thus, based on both SDN and distributed protocols). As an example, in [37], a recovery mechanism is proposed in an environment where legacy network devices and SDN switches co-exist. In [38], a centralized controller is used for long-term optimization relying on OpenFlow to configure arbitrary forwarding paths during normal operations, while distributed protocols (e.g., Interior Gateway Protocol) can be used for short-term reaction to failures. Hybrid SDN may provide more robustness against failures than classical SDN. In our paper, the proposed solution does not exploit legacy-network distributed protocols, while it operates with the only NETCONF/YANG in both normal and failure conditions.

A different approach in order to enhance reliability is to rely on redundant transmissions. IEEE 802.1Qbd amendment the fully distributed case is investigated where the resource allocation and registration is made in a distributed fashion, while the endpoints only interact with the access TSN bridge over the UNI.
cases, the computational requirements can be complex and may require a detailed knowledge of the application software/hardware within each end station. The fully centralized architecture includes two main control plane components: the Centralized User Configuration (CUC) and the Centralized Network Configuration (CNC). CUC discovers end stations, retrieves end station capabilities and user requirements, and configures TSN features in end stations. In this paper, the implementation of CUC is out of scope and we assume end station discovery already performed and end stations configured by CNC. CUC features can be also assumed to be integrated into CNC. CNC acts as the SDN controller. In general, CNC uses remote management to discover physical topology, to retrieve bridge capabilities, and to configure TSN features/resources in each bridge. Regarding TSN resource management, the reader may refer to IEEE802.1Qbv, which includes the Management Information Base (MIB) for support of the Scheduled Traffic Enhancements for 802.1Q Bridges. In the case of our implementation, as it will be described in Sec. V, we will refer to TAPRIO which implements a simplified version of IEEE802.1Qbv. The next subsections will describe the control modules involved in our proposed solution and the FSM YANG model.

The main functional building blocks are detailed in the following.

**SDN controller**: it is responsible for traditional network configuration and monitoring. Thus, the SDN controller configures bridges in normal network conditions. Moreover, the SDN controller installs FSMs as it will be detailed next. The protocol considered between the SDN controller and the TSN data plane is NETCONF. The SDN controller configures each interested device at the data plane. To this purpose, it sends a NETCONF <edit-config> message towards each device agent. The content of this message is based on a specific YANG data model describing the configuration parameters of the device and includes the values of these parameters should take. The NETCONF <edit-config> message writes such values in the local NETCONF Server at the agent. In our scheme, the SDN controller writes in the NETCONF Server also the description of the FSM that each agent should consider for delegation in the case of specific events. We assume all the control plane messages sent in an out-of-band

---

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
control plane.

Then, each TSN node presents modules for local control: the Agent and the TSN App.

**Agent:** It is the local controller of each data plane device. The Agent has access to the local NETCONF Server, it parses configuration parameters and performs device configuration accordingly. Moreover, the Agent is also responsible to parse and interpret local FSMs and, if requested by network events, to perform the corresponding set of actions. The Agent handles layer-2 and layer-3 operations and interacts with the TSN App. With the proposed solution, control plane messages are exchanged also between different agents. Even those control plane messages are sent in the out-of-band control plane. More details on the Agent will be provided next.

**TSN App:** It is responsible for the actual configuration of the TSN parameters in the bridge. TSN configuration can be made either from a NETCONF call by the SDN controller or locally by the Agent, because of a F8M state transition. No logic resides on the TSN App, it is only used to execute the proper configuration.

### B. YANG MODEL TO INSTALL FSM

FSMs are described by YANG [13]. The related YANG tree is depicted in Fig. 2 and here described.

The attribute `<current-state>` defines the current state of the FSM. Then, the YANG tree presents a list of states, where each `<state>` is associated to an identifier (`<id>`) and to a string of description (`<description>`). Each state presents a list of transitions to other states, where each `<transition>` is associated to a name, a type (taken from a pool of possible transition types predefined inside the YANG model), a string of description, and other attributes. Among them, the attribute `<filter>` enables to further express a transition (e.g., when a transition is triggered by a parameter exceeding a threshold, as a monitored packet loss exceeding a threshold).

Finally, a list of actions is described. Each `<action>` is defined by an identifier (`<id>`) and a type (`<type>`). An action can be `<conditional>` or `<simple>`. The former is executed depending on the check of specific conditions described through the attribute `<statement>`, while the latter is directly executed. An action can be taken locally or toward a remote node. In the second case, the `<remote-address>` attribute is needed to specify which node has to be involved. The `<execute>` attribute actually recalls the execution of the action. If more actions have to be executed, these actions can be executed sequentially according to the `<next-action>` attribute. Finally, the `<next-state>` attribute defines the new state of FSM after that transition.

### IV. A USE CASE: FAILURE RECOVERY WITH DRTSN

The proposed recovery scheme – DRTSN – is used to realize recovery after failures without involving the SDN controller. However, a similar technique can also be applied in the case of performance degradation. Thus, several “Failure” states can be considered, each one associated to a specific failure (e.g., port down) or performance degradation (e.g., experienced delay above a given threshold).

The SDN controller pre-installs FSMs at each data plane device in order to affect data plane operations in the case of a failure, such as frame rerouting according to a pre-computed path, buffer activation, scheduling, class of service redefinition. An illustrative example is depicted in Fig. 3, where a stream of frames is considered between a talker-listener pair along the path A-E-D. Our goal is to instruct data plane devices to perform rerouting without SDN controller intervention upon fault in order to speed up recovery.

**During Normal operation:** The NETCONF `<edit-config>` message installs FSMs. The content of these messages is based on the YANG model described in Sec. III-B. Upon reception of the `<edit-config>` messages, data plane devices are instructed on the actions to perform based on the network state.

FSM can be dynamically updated based on traffic load, thus, as an example, under stable conditions, the pre-computed backup path of a stream can be changed by the SDN controller based on buffers load. For the case of hard failure, the installed FSMs are abstracted in Fig. 4 and are composed by two states, “Stable”, “Failure”, each one associated to an identifier as reported in the YANG model of Fig. 2. At the “Stable” state, devices operate according to the original configurations acted by the SDN controller. Reconfigurations are taken at the “Failure” state as described in the following.

**Failure event handling:** Once a failure is detected by a node (e.g., through loss of frame), a state transition of the FSM at this node is taken, moving the state to “Failure”, as shown in Fig. 4. Following the example in Fig. 3, the link between nodes E and D fails. Assuming D detecting the failure, the FSM at D moves to “Failure” state. At this state, some of the actions summarized in Tab. 1 are executed. In particular, the node detecting the failure performs the following main action: an ad-hoc defined `<rpc>` message is sent to each node of the pre-computed backup path to trigger its state transition. Such `<rpc>` simply changes the FSM `<current-state>` value to the “Failure” state at the interested nodes (A-B-C-D in Fig. 3). The content of the `<rpc>` is shown in Fig. 5 and consists of an XML including the new value of the `<current-state>` attribute in the FSM YANG model. If the node detecting the failure is the destination node, further actions must be taken. In this work, layer-3 rerouting is assumed. The following actions are then considered: add a new entry in the local routing table in order to receive frames from the backup path; reconfigure TSN buffers. Finally, to address synchronization issues, the node may inform, through a NETCONF `<notification>` message, the SDN controller that the recovery action has been taken.

Each node receiving the `<rpc>` message (from the node that detected the failure) changes its FSM state and takes actions at the data plane according to the information stored in its local FSM. Actions involve: adding new entries in the local routing table and TSN buffers reconfiguration. This
second operation consists in calling the TSN app with proper configuration parameters. The FSM could also encompass other actions such as the change of traffic priority.

Finally, it has to be mentioned that after the traffic is rerouted along a backup path, the SDN controller may precompute another backup path (together with TSN configurations) and install the related FSMs to provide robustness against several failures. If a failure occurs while FSMs are not installed – and thus DRTSN cannot be applied – the agent of the node detecting the failure should inform the SDN controller, then it sends an alarm (implemented with a NETCONF <notification> message) to the SDN controller, which will perform classical reactive restoration.

V. IMPLEMENTATION IN LINUX

In this subsection the implementation of the main control modules and data plane involved during delegated restoration is presented.

A. DATA PLANE

The data plane has been realized by connecting different Ubuntu 20.04 machines (from 2 to 4 physical PCs). Each machine has been configured to act as a router by enabling the IP forwarding and installing the routes to let the traffic flowing along the primary paths. TSN has been exploited through TAPRIO qdisc, implementing a simplified version of the scheduling defined by IEEE 802.1Qbv [16]. TAPRIO has been configured on the egress interface of each machine and iptables classifier rules have been installed. Referring to Fig. 6, as a configuration example, in Switch-1 we configured the interface enp4s0f1 to use TAPRIO. Such configuration is acted by the TSN app, whose implementation is described next. The iptables command is launched to set the priority against several failures. If a failure occurs while FSMs are not installed – and thus DRTSN cannot be applied – the agent of the node detecting the failure should inform the SDN controller, then it sends an alarm (implemented with a NETCONF <notification> message) to the SDN controller, which will perform classical reactive restoration.

V. IMPLEMENTATION IN LINUX

In this subsection the implementation of the main control modules and data plane involved during delegated restoration is presented.

A. DATA PLANE

The data plane has been realized by connecting different Ubuntu 20.04 machines (from 2 to 4 physical PCs). Each machine has been configured to act as a router by enabling the IP forwarding and installing the routes to let the traffic flowing along the primary paths. TSN has been exploited through TAPRIO qdisc, implementing a simplified version of the scheduling defined by IEEE 802.1Qbv [16]. TAPRIO has been configured on the egress interface of each machine and iptables classifier rules have been installed. Referring to Fig. 6, as a configuration example, in Switch-1 we configured the interface enp4s0f1 to use TAPRIO. Such configuration is acted by the TSN app, whose implementation is described next. The iptables command is launched to set the priority against several failures. If a failure occurs while FSMs are not installed – and thus DRTSN cannot be applied – the agent of the node detecting the failure should inform the SDN controller, then it sends an alarm (implemented with a NETCONF <notification> message) to the SDN controller, which will perform classical reactive restoration.
FIGURE 6. Data plane and control plane with 4 nodes

module: tsn-taprio
   +--rw tsn-taprio-structure
      +--rw interface* [dev]
         |   +--rw dev string
         |   +--rw parent? string
         |   +--rw handle? uint32
         |   +--rw num-tc? uint8
         |   +--rw map? string
         |   |   +--rw queue* [id]
         |   |      |   +--rw id uint32
         |   |      |   +--rw elem? string
         |   |   +--rw base-time? uint64
         |   +--rw clockid? string
         |   +--rw sched-entries
         |      +--rw sched-entry* [id]
         |         |   +--rw id uint32
         |         +--rw command? string
         |         +--rw gatemask? string
         |         +--rw interval? uint64

rpcs:
   +---x taprio-set
      +----w input
      |   +--w command? string
      +--ro output
      +--ro result? boolean

FIGURE 7. TAPRIO module

field to the forwarded traffic according to the traffic class defined by TAPRIO. The following command assigns an highest priority to the UDP traffic that has as destination port 6666:

```
iptables -t mangle -A POSTROUTING -p udp --dport 6666 -j CLASSIFY --set-class 0:1
```

Lowest priority is assigned to the traffic that has as UDP destination port 7777:

```
iptables -t mangle -A POSTROUTING -p udp --dport 7777 -j CLASSIFY --set-class 0:0
```

Each data plane node is locally controlled through an agent. The implementation of the following control modules is described: TSN app, FSM in the NETCONF Server, and the FSM agent.

B. TSN APP

The TSN app is implemented in Python and applies configuration to TAPRIO qdisc. The TSN app has access to the NETCONF server (whose implementation is based on Netopeer). The TSN app parses the XML TAPRIO configuration stored in the NETCONF Server and issues the related command. In this case, the TSN app reads the TAPRIO configuration
Actions to be executed at the “Failure” state according to “rdnt” digits = 1101

- **r digit = 1**: <rpc> sent to the agents with the following <remote-address>: 192.168.255.129, and 192.168.255.4 changing local FSM state to 002 value
- **d digit = 1**: Change of local routing table to accept interrupted traffic from the backup path port
- **n digit = 0**: no <notification> message to the SDN controller
- **t digit = 1**: reconfigure local TSN buffers on the ens4 interface with parameters as in Fig. 9

<tsn-taprio-structure xmlns="http://sssa.it/yang/tsn-taprio">
   <interface>
      <dev>ens4</dev>
      <parent>root</parent>
      <handle>10</handle>
      <num-tc>2</num-tc>
      <map>1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1</map>
      <queues>
         <queue>
            <id>0</id>
            <elem>1@0</elem>
         </queue>
         <queue>
            <id>1</id>
            <elem>1@1</elem>
         </queue>
      </queues>
      <base-time>1528743495910289987</base-time>
      <clockid>CLOCK_TAI</clockid>
      <sched-entries>
         <sched-entry>
            <id>0</id>
            <command>5</command>
            <gatemask>01</gatemask>
            <interval>800000</interval>
         </sched-entry>
         <sched-entry>
            <id>1</id>
            <command>5</command>
            <gatemask>02</gatemask>
            <interval>200000</interval>
         </sched-entry>
      </sched-entries>
   </interface>
</tsn-taprio-structure>
parameters set by the SDN controller through the `<edit-config>` message, as it happens during normal operation on the primary path. Alternatively, the TSN app can be invoked to read the TAPRIO configuration from the specific FSM state. In the latter case, the TSN app also updates the NETCONF Server based on the new TAPRIO configuration.

The view of the TAPRIO module is shown in Fig. 7. The parameters described by the YANG model of Fig. 7 are:

- `interface`: it represents the list of interfaces at the node, each one identified by the string `dev`
- `dev`: the name of the interface
- `parent`: class id. Default: root
- `handle`: unique handle identifier for qdisc
- `num-tc`: number of traffic classes. Max 16;
- `maps`: list to map priority to traffic classes;
- `queues`: identifies a list of queues, where each element is identified by `elem`:
  - `mapping`: The format is `queue_countqueue_offset`;
  - `base-time`: reference time in nanoseconds to start the schedule;
  - `clock-id`: reference clock. Default: CLOCK_TAI
  - `sched-entries`: list of sched-entries
  - `id`: to identify the element in the sched-entries list;
  - `command`: the only supported `<command>` is “S”, which means “SetGateStates”;
  - `gatemask`: a bitmask where each bit is associated with a traffic class;
  - `interval`: time duration in nanoseconds specifying for how long the sched-entry should be held before moving to the next one.

### C. FSM INSTALLED IN THE NETCONF SERVER

The NETCONF server is based on Netopeer and stores current configuration parameters (e.g., running TSN configuration) and FSMs. Fig. 8(a) shows part of the FSM installed at a destination node when it identifies a failure in an incoming port. Fig. 8b summarizes the actions to be locally executed based on the XML at the “Failure” state. The actions are stored in the `<execute>` attribute, which includes the following fields spaced by `::`

- `<elem>`: a mask; ii) the state to be configured in remote nodes. Then, the `<execute>` attribute also includes the XML configuration of TAPRIO reflexing the YANG model of Fig. 7 (for simplicity an example is shown in a separate figure, Fig. 9). The mask appears as a set of `rdst` digits, as follows:
  - “r”: if equal to 1, an `<rpc>` has to be sent to at least a remote node; 0 otherwise
  - “d”: if equal to 1, the communication with the data plane node in order to change the local routing table is enabled; 0 otherwise
  - “n”: if equal to 1, the `<notification>` message to the SDN controller is enabled; 0 otherwise
  - “t”: if equal to 1, TSN buffers reconfiguration is locally enabled; 0 otherwise

Thus, in the reported XML, the considered agent should perform the following actions, as summarized in Fig. 8b. The digit `r` set to 1 implies that `<rpc>` messages must be sent to agents with address identified by the `<remote-address>` attribute, thus to the following agents: 192.168.255.129 and 192.168.255.4. The remote state set by the `<rpc>` in the current example has identifier `002`, i.e., “Failure” state. The digit `d` set to 1 drives a reconfiguration of the local routing table on the pre-installed entry associated to the backup path. Considering L3 forwarding, the SDN controller pre-installed in the routing table the backup route with an high-metric value (30 in our implementation), this way only the primary route (which has a lower metric value of 20) is exploited before the failure. At the moment of the failure, the metric of the backup route is set to a low value (10 in our implementation) in order to preempt the primary route. Then, the TSN app is called with XML TAPRIO configuration parameters, as shown in Fig. 9, with values associated to the attributes of the YANG model in Fig. 7.

1) **FSM agent**

The node agent is implemented in a docker container and it is composed of different frameworks used by the Netopeer software to implement the agent. In order interact with the NETCONF server we have developed a Sysrepo plugin that implements the actions related to the state change in the FSM YANG model. The implemented agent is shown in Fig. 10, connected with the TSN switch at the data plane. At the data plane, once a failure is detected, a Python script (Failure Detector in the figure) sends a message over a socket to notify the local agent about the failure. At the agent, a Listener implemented in Python is responsible to receive and process the failure notification. Once the Listener is notified about the failure, it invokes the Sysrepo rpc_fsm caller to trigger the FSM parsing based on the failure. A Sysrepo rpc_fsm handler receives the RPC request, parses it, and retrieves the information associated to the new state in the configured FSM. The information in the FSM is described in the previous subsection including `<rpc>` messages sent to remote nodes and local data plane reconfiguration. Regarding the latter action a Sender implemented in Python may invoke the TSN app and may add a new entry in the local routing table.

### VI. PERFORMANCE EVALUATION

Performance of DRTSN is compared with a fully centralized recovery (CR) in an experimental testbed. According to CR, once a failure is detected, a NETCONF `<notification>` message (used as an alarm) is sent to the SDN controller (implemented based on ONOS [41]). The SDN controller processes the received alarm and then acts the reconfiguration of the traffic interested in the failure. Several experiments have been carried out for performance evaluation. Three network topologies have been tested: a 2-nodes ring, a 3-nodes ring, and a 4-nodes meshed network (represented in Fig. 6). The characteristics of each PCs are reported in table 2. Each network node is a physical machine emulating...
a TSN switch with TAPRIO enabled at each Ethernet port. Moreover, each physical machine runs the dockerized node agent. Control plane communications between node agents, and between node agents and SDN controller are done in an out-of-band control plane network. Thus, exchanged control plane messages (e.g., <rpc> messages during recovery with DRTSN) do not affect the traffic in the data plane. Traffic is injected in the network through a Spirent traffic generator/analyzer [42] enabling time performance measurements. Each constant bit rate stream is composed of frames of 128 Bytes generated with a rate of 10000 frames/second. Single-link failures are generated forcing a node port down. Recovery delay is measured, defined as the time between the failure and the traffic is rerouted on the recovery path.

Finally, DRTSN is also compared with FRER in terms of the average number of frames per queue.

### A. DATA PLANE PERFORMANCE

This section presents the performed measurements at the data plane.

1) **Experiment 0**: the proof of concept is demonstrated with the higher- and the lower-priority streams (as described in Sec. V-A) on the 4-nodes mesh topology (Fig. 6). The failure is generated on port enp4s0f1 of Switch-1, then the streams are recovered with DRTSN and CR. In average, with DRTSN, traffic is recovered in 823 ms, while with CR in 5171 ms. DRTSN permits...
TABLE 3. Intra-node time contributions upon failure at the node detecting the failure

| Operation          | Required time [ms] |
|--------------------|--------------------|
| FSM parsing        | 1                  |
| <rpc> generation   | 325                |
| TSN app execution  | 20                 |
| Route adding       | 6.32               |

The main time contribution is required by the generation of \(<\text{rpc}>\) messages triggering FSM state transition are sent sequentially and not in parallel. Indeed, in the case of 2-nodes topology, the recovery path is composed of one hop and the node detecting the failure has to send only one remote \(<\text{rpc}>\) message. In the case of the 3- and 4-nodes topologies, the backup route is always of two hops, thus the node detecting the failure sends two remote \(<\text{rpc}>\) messages. This sentence is supported also by the intra-node time contributions analysis presented in Sec. VI-B.

3) Experiment 2: recovery delay is measured at varying the number of network nodes. 2-nodes and 3-nodes ring topologies, and the 4-nodes meshed topology have been tested, with higher and lower priority streams (as in Sec. V-A) present in the network. Routing for the primary path is performed on the shortest path in terms of hops, while for the backup path on the shortest path which is link-disjoint from the primary. Fig. 11 shows the recovery delay versus the number of network nodes.

B. INTRA-NODE TIME CONTRIBUTIONS AND CONTROL PLANE OVERHEAD

The time contributions to perform the agent operations with DRTSN – which also have an impact on the recovery delay – have been analyzed and summarized in Tab. 3. After a failure, the node agent requires around 1 ms to parse FSM. The main time contribution is required by the generation of \(<\text{rpc}>\) message to a remote node, which is around 325 ms. This delay includes the following contributions: i) the time to establish an ssh session between the agent and the agent at the remote node; ii) the time to establish NETCONF over ssh; iii) the time for transmitting the \(<\text{rpc}>\) message. The \(<\text{rpc}>\) generation time is also comparable with the difference between the recovery delay achieved in the 2-nodes (520 ms) and in the 3-nodes (846 ms) topologies, where one \(<\text{rpc}>\) and two \(<\text{rpc}>\) messages are sent to remote nodes, respectively. The TSN app execution requires 20 ms. This time contribution includes the reconfiguration of TAPRIO and also the updating of the NETCONF server based on the new TAPRIO configuration. Finally, the addition of a new entry on the local routing table takes around 6.32 ms.

With DRTSN, upon failure, \(<\text{rpc}>\) messages to remote nodes are exchanged. In particular, an \(<\text{rpc}>\) is generated for each remote node of the pre-computed backup path, excluding the node detecting the failure if present in the backup path. With CR, more control plane messages are exchanged. First, a NETCONF \(<\text{notification}>\) messages is sent from the node detecting the failure to the SDN controller to inform it about the failure. Then, the NETCONF \(<\text{edit-config}>\) messages are sent to each node in the backup path to reconfigure it upon failure.
C. OPTIMIZATION OF DRTSN

By observing Tab. 3, the main bottleneck is due to the \texttt{<rpc\>}
generation time contribution, which includes the time to open
a ssh session, a NETCONF session, and finally the time to
generate the \texttt{<rpc\>}. Thus, we made other tests on recovery
delay by having already open NETCONF sessions (and ssh as
well) between the node detecting the failure and the remote
agents. Then, at the time of recovery, the \texttt{<rpc\>} to change
FSM state on remote agents can be directly sent because
the ssh session and NETCONF over ssh are already open.
Results are shown in Table 4 in comparison with the DRTSN
as in Sec. 5.1 (requiring to open ssh and NETCONF sessions
before sending \texttt{<rpc\>} messages). The considered topology
is the 4-nodes meshed one as in Fig. 6.

This optimization of DRTSN permits to reduce the \texttt{<rpc\>}
generation time from 325 ms to 7 ms. This results in an
overall recovery delay decreased by a factor higher than three
(from 823 ms to 254 ms), which includes the generation of two
\texttt{<rpc\>} messages.

D. COMPARISON BETWEEN DRTSN AND FRER

To compare DRTSN, CR, and FRER a custom-built simulator
has been utilized, given that current TAPRIO implementation
does not support FRER. The simulator is event driven and
it is written in C++. The events are organized in a Binary
Heap Tree [43] and the flow of the simulation is driven by
events such as traffic streams arrival, frame transmission,
l link failures, control packet generation and transmission.
A ring-topology with ten nodes and 100-Mb/s interfaces has
been assumed. Traffic streams arrive following a Poisson
distribution with an average inter-arrival time of 1/\lambda. Bursts
of 100 frames per stream are considered. Single-link failures
are randomly generated among the network links. We con-
sidered that queues are long enough to avoid packet loss in
the buffers. For FRER, we considered the implementation
where two Member Streams are generated following two
disjoint routes. For FRER, the secondary path is computed
as the shortest path that is link disjoint from the primary
one. DRTSN, CR, and FRER are compared in terms of the
average number of frames per queue during normal network
operations. Results are plotted with a confidence interval of
95\% at varying 1/\lambda.

Fig. 13 shows the average number of frames in a queue
versus the mean inter-arrival time of streams for all the three
schemes: DRTSN, FRER, and CR. The figure shows that
FRER may create a very high load in the network due to
the redundant transmission with respect to DRTSN and CR.
Moreover, note that the slope of the curve associated to FRER
is much larger than the ones associated to FSM and CR. This
is due to the fact that, in a ring topology, with FRER, a stream
(considering the two member streams of a compound stream)
always use all the nodes, thus highly loading the network.

VII. CONCLUSIONS AND FUTURE WORK

The paper proposed a new mechanism for managing reliability
in SDN-controlled TSN networks with NETCONF. Finite
State Machines (FSMs) permit to proactively instruct the
nodes on the actions to perform in case of specific events such
as failures or performance degradation. If an event occurs, the
SDN controller is by-passed thus reducing recovery time. A
failure recovery use case has been analyzed providing details
of the implementation and of the performance evaluation.
Measurements in an experimental testbed have shown that
the proposed method strongly reduces recovery time with
respect to a reactive restoration. Simulations have shown that
the proposed method avoids to overload the network with
respect to FRER. However, our method is not proposed to
replace FRER: the FSM-based delegation can coexist in the
network with FRER, e.g., for specific service classes.

Future works will investigate the optimization of the pro-
posed method. Based on the observations in the experimen-
tal testbed we have seen that the main time contribution
to the recovery delay is due to the generation of \texttt{<rpc\>}
messages since they require to open an ssh session, to open
a NETCONF session, and then to generate the message. The
possibility of keeping already open sessions before \texttt{<rpc\>}
delivery will be investigated to further reduce recovery delay.
Then, future work will investigate the potentials of integrat-
ing FSM and P4. Other possible studies may investigate
the trade off between the number of pre-installed backup
routes (and related TSN features) and calculations/memory
needed, with the objective of guaranteeing robustness against
multiple faults.

REFERENCES

[1] “Time-sensitive networking (tsn) task group,” [Online]. Available:
https://1.ieee802.org/tsn/
[2] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodol-
molky, and S. Uhlig, “Software-defined networking: A comprehensive
survey,” Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, 2015.
[3] M. Karakus and A. Durresi, “A survey: Control plane scalability
issues and approaches in software-defined networking (sdn),” Computer
Networks, vol. 112, pp. 279 – 293, 2017, [Online]. Available:
http://www.sciencedirect.com/science/article/pii/S138912861630411X
[4] G. Saldamlı, H. Mısha, N. Ravi, R. R. Kodati, S. A. Kuntamukkala, and
L. Tawalbeh, “Improving link failure recovery and congestion control
in sdns,” in 2019 10th International Conference on Information and Commu-
nication Systems (ICICS), 2019, pp. 30–35.
[5] “Frame replication and elimination for reliability,” ISO/IEC/IEEE 8802-
1CB, 2019.
[6] F. von Tüllenburg Thomas and Th. Pfiefenberger, “Layer-2 failure recovery
methods in critical communication network,” in Proc. of ICNS2016, 2016.
[7] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi,
“Openflow-based segment protection in ethernet networks,” IEEE/OSA
Journal of Optical Communications and Networking, vol. 5, no. 9, pp.
1066–1075, 2013.
[8] R. Emms, M. Björklund, J. Schoenwaelder, and A. Bierman, “RFC 6241,
network configuration protocol (NETCONF),” Internet Engineering Task
Force (IETF), June, 2011.
[9] M. Björklund, “YANG - a data modeling language for the network configu-
ration protocol (NETCONF),” IETF RFC 6020.
[10] S. Brooks and E. Udulag, “Time-sensitive networking: From theory to
implementation in industrial automation,” Tech. Rep.
[11] S. Kehrer and F. Chen, “Tsn configuration – focusing on network man-
agement protocols,” Tech. Rep. [Online]. Available: https://www.ieee802.
org/1/files/public/doc/2017/ce-kehrer-tsn-configuration-0317-v01.pdf
[12] “Ttech releases world’s first vendor-independent tsn configuration soft-
ware,” Tech. Rep. [Online]. Available: https://www.tttech.com/
ttech-releases-worlds-first-vendor-independent-tsn-configuration-software/
[13] N. Sambo, P. Castoldi, G. Fioccola, F. Cugini, H. Song, and T. Zhou, “YANG model for finite state machine,” draft-sambo-netmod-yang_fsm-05, May 2019.

[14] M. Dallaglio, N. Sambo, F. Cugini, and P. Castoldi, “Method for managing a telecommunication network,” in Patent No.: US 10,749,624 B2, Aug. 18, 2020.

[15] N. Sambo, “Locally automated restoration in sdn disaggregated networks,” IEEE/OSA Journal of Optical Communications and Networking, vol. 12, no. 6, pp. C23–C30, 2020.

[16] “Enhancements for scheduled traffic,” IEEE 802.1Qqv, 2016.

[17] “IEEE standard for ethernet amendment 5: Specification and management parameters for interspersing express traffic,” IEEE 802.3br-2016, 2016.

[18] “Frame preemption,” IEEE 802.1Qbu, 2015.

[19] “IEEE standard for local and metropolitan area networks–bridges and bridged networks – amendment 34: Asynchronous traffic shaping,” IEEE 802.1Qer-2020, 2020.

[20] “Cyclic queuing and forwarding,” P802.1Qch, 2015.

[21] A. Nasrallah et al., “Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research,” IEEE Communications Surveys Tutorials, vol. 21, no. 2, pp. 138–145, 2019.

[22] W. Steiner, S. S. Cricunias, and R. Serna Oliver, “Traffic Planning for Time-Sensitive Communication,” IEEE Communications Standards Magazine, vol. 2, no. 2, pp. 42–47, 2018.

[23] S. S. Cricunias, R. Serna Oliver, M. Chmelik, and W. Steiner, “Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks,” in RTNS, ACM, 2016, p. 183–192.

[24] L. Zhao, P. Pop, and S. S. Cricunias, “Worst-Case Latency Analysis for IEEE 802.1 Qbv Time Sensitive Networks Using Network Calculus,” IEEE Access, vol. 6, pp. 41 803–41 815, 2018.

[25] D. Maxim and Y.-Q. Song, “Delay analysis of AVB traffic in time-sensitive networks (TSN),” in RTNS, 2017, pp. 18–27.

[26] N. Reusch, L. Zhao, S. S. Cricunias, and P. Pop, “Window-Based Scheduling Synthesis for Industrial IEEE 802.1Qbv TSN Networks,” in 2020 16th IEEE International Conference on Factory Communication Systems (WFCS), 2020, pp. 1–4.

[27] S. M. Lautersen, P. Pop, and W. Steiner, “Routing optimization of AVB streams in TSN networks,” ACM Sigbed, vol. 13, no. 4, pp. 43–48, 2016.

[28] N. G. Nayak, F. Duerr, and K. Rothermel, “Routing algorithms for IEEE 802.1 Qbv networks,” ACM SIGBED, vol. 15, no. 3, pp. 13–18, 2018.

[29] N. G. Nayak, F. Durr, and K. Rothermel, “Time-sensitive software-defined network (TSSDN) for real-time applications,” in RTNS, 2016.

[30] V. Gavrilut, L. Zhao, M. L. Raagaard, and P. Pop, “AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN,” IEEE Access, vol. 6, pp. 75 229–75 243, 2018.

[31] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Müh, “ILP-based joint routing and scheduling for time-triggered networks,” in RTNS, 2017, pp. 8–17.

[32] M. Pahlevan, N. Tabassam, and R. Obermaisser, “Heuristic list scheduler for time triggered traffic in time sensitive networks,” ACM Sigbed, vol. 16, no. 1, pp. 15–20, 2019.

[33] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, “Enabling fog computing for industrial automation through Time-Sensitive Networking (TSN),” IEEE Communications Standards Magazine, vol. 2, 2018.

[34] C. Mas-Machuca, F. Musumeci, P. Vizarreta, D. Pezaros, S. Jouet, M. Tornatore, A. Hmaisy, M. Liyanage, A. Kurtov, and A. Bracken, Reliable Control and Data Planes for Softwarized Networks. Cham: Springer International Publishing, 2020, pp. 243–270.

[35] G. Bianchi, S. Pontarelli, M. Bonola, C. Cascone, D. Sanvito, and A. Capone, “Method of handling data packets through a conditional state transition table and apparatus using the same,” Jul. 7 2020, US Patent 10,708,179.

[36] R. Amin, M. Reisslein, and N. Shah, “Hybrid sdn networks: A survey of existing approaches,” IEEE Communications Surveys Tutorials, vol. 20, no. 4, pp. 3259–3306, 2018.

[37] C.-Y. Chu, K. Xi, M. Luo, and H. J. Chao, “Congestion-aware single link failure recovery in hybrid sdn networks,” in 2015 IEEE Conference on Computer Communications (INFOCOM), 2015, pp. 1086–1094.

[38] O. Tilmans and S. Vissicchio, “Igp-as-a-backup for robust sdn networks,” in 10th International Conference on Network and Service Management (CNSM) and Workshop, 2014, pp. 127–135.

[39] 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, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[40] “Bridges and bridged networks - amendment 31: Stream reservation protocol (SRP) enhancements and performance improvements,” IEEE Std 802.1Qec, 2018.

[41] “ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out,” https://onosproject.org.

[42] https://www.spirent.com/products/aion.

[43] J. Williams, “Algorithm 232 – heapsort,” Communications of the ACM, vol. 7, no. 6, pp. 347–348, 1964.

### VOLUME 4, 2016