IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received September 9, 2021, accepted September 28, 2021, date of publication September 29, 2021,
date of current version October 11, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3116880

Enabling Delegation of Control Plane Functionalities for Time Sensitive Networks

NICOLA SAMBO[1], SILVIA FICHERA[1], ANDREA SGAMBELLURI[1], GIUSEPPE FIOCCOLA[2], PIERO CASTOLDI[1], (Senior Member, IEEE), AND KOSTAS KATSALIS[2]

[1]Scuola Superiore Sant'Anna, 56124 Pisa, Italy
[2]Huawei Technologies Duesseldorf Gmbh, 80993 Munich, Germany

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

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 event occurs, 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], that are considered and already supported in TSN networks [10]–[12], especially to offer open and vendor-independent network configuration [10], [12].

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

The associate editor coordinating the review of this manuscript and approving it for publication was Fung Po Tso.

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 Queuing 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 queueing 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], [23], 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 are proposed in [31]. A List Scheduling-based heuristic and scheduling and routing are 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 802.1Qdj. 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 802.1Qdd amendment, the fully distributed case is investigated, while in the Centralized Network-Distributed User Model, the UNI exists between the endpoints and the access TSN bridge; however, in the latter, the flow specs are passed to a CNC which is responsible for the network segment. In 802.1Qdd 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 802.1CB specifies the procedure for FRER in a redundant transmission for reliability purposes [5]. FRER assumes the existence of a Talker-Listener (i.e., source/destination) pair per data stream. A stream is composed of a number of packets transmitted per time interval. FRER replicates a stream (thus, the packets) into copies named *member streams*, which follow alternative routes along the network. Multiple member streams compose a *compound stream*. Components receiving multiple copies of the same packet proceed with the elimination of the replicas. As an example, a talker may be the only element generating two member streams from a stream. In this case, the listener eliminates replicas of the same packets. According to 802.1CB [5], bridges can replicates the packets of the stream, splitting the copies into the multiple member streams, and then rejoins those member streams at one or more other points, eliminates the replicates, and delivers the reconstituted stream from those points. FRER requires the management of the following two main parameters: a) *stream_handle* which is an integer identifying the compound stream to which the packet belongs and b) *sequence_number*, an unsigned integer identifying the order in which the packet was transmitted relative to other packets in the same compound stream.

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, 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

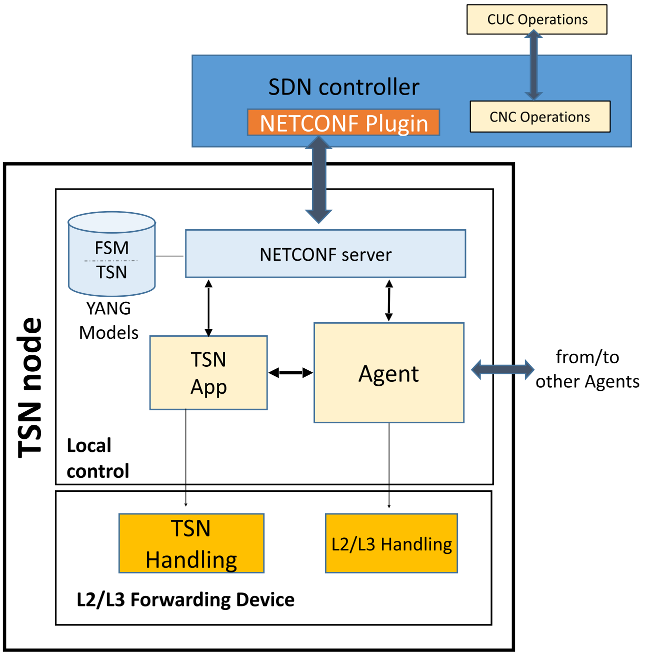


FIGURE 1. System architecture.

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 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 FSM 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.

```
+--rw current-state?   state-id-type
+--rw states
   +--rw state [id]
      +--rw id             state-id-type
      +--rw description?   string
      +--rw transitions
         +--rw transition [name]
            +--rw name           string
            +--rw type?          string
            +--rw description?   string
            +--rw filters
            |  +--rw filter [filter-id]
            |     +--rw filter-id    uint32
            +--rw actions
            |  +--rw action [id]
            |     +--rw id             transition-id-type
            |     +--rw type           enumeration
            |     +--rw conditional
            |     |  +--rw statement    string
            |     |  +--rw true
            |     |  |  +--rw remote-address?   inet:ip-address
            |     |  |  +--rw execute
            |     |  |  +--rw next-action?      transition-id-type
            |     |  +--rw false
            |     |     +--rw remote-address?   inet:ip-address
            |     |     +--rw execute
            |     |     +--rw next-action?      transition-id-type
            |     +--rw simple
            |        +--rw remote-address?   inet:ip-address
            |        +--rw execute
            |        +--rw next-action?      transition-id-type
            +--rw next-state?    state-id-type
```

FIGURE 2. Yang tree of FSM.

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.

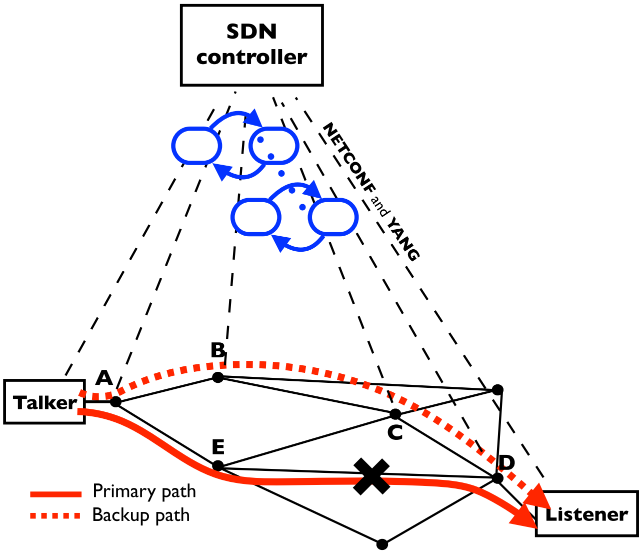


FIGURE 3. Network and FSM installation.

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.

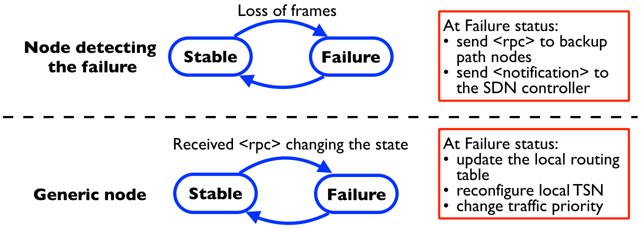

Loss of frames
Node detecting the failure
Stable
Failure
At Failure status:
• send <rpc> to backup path nodes
• send <notification> to the SDN controller
Received <rpc> changing the state
Generic node
Stable
Failure
At Failure status:
• update the local routing table
• reconfigure local TSN
• change traffic priority

FIGURE 4. FSMs installed in the nodes.

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: update

TABLE 1. Actions for DRTSN.

Action
Send <rpc> to the nodes along the backup route to trigger FSM state transition
Send <notification> to the SDN controller to inform it about the failure
Update routing table
Configure TSN buffers
Change traffic priority

```
<finite-state-machine xmlns="http://sssup.it/fsm">
   <current-state>2</current-state>
</finite-state-machine>]]>
```

FIGURE 5. <rpc> triggering state transition.

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: updating of 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 pre-compute 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 field to the forwarded traffic according to the traffic class defined by TAPRIO. The following command assigns an

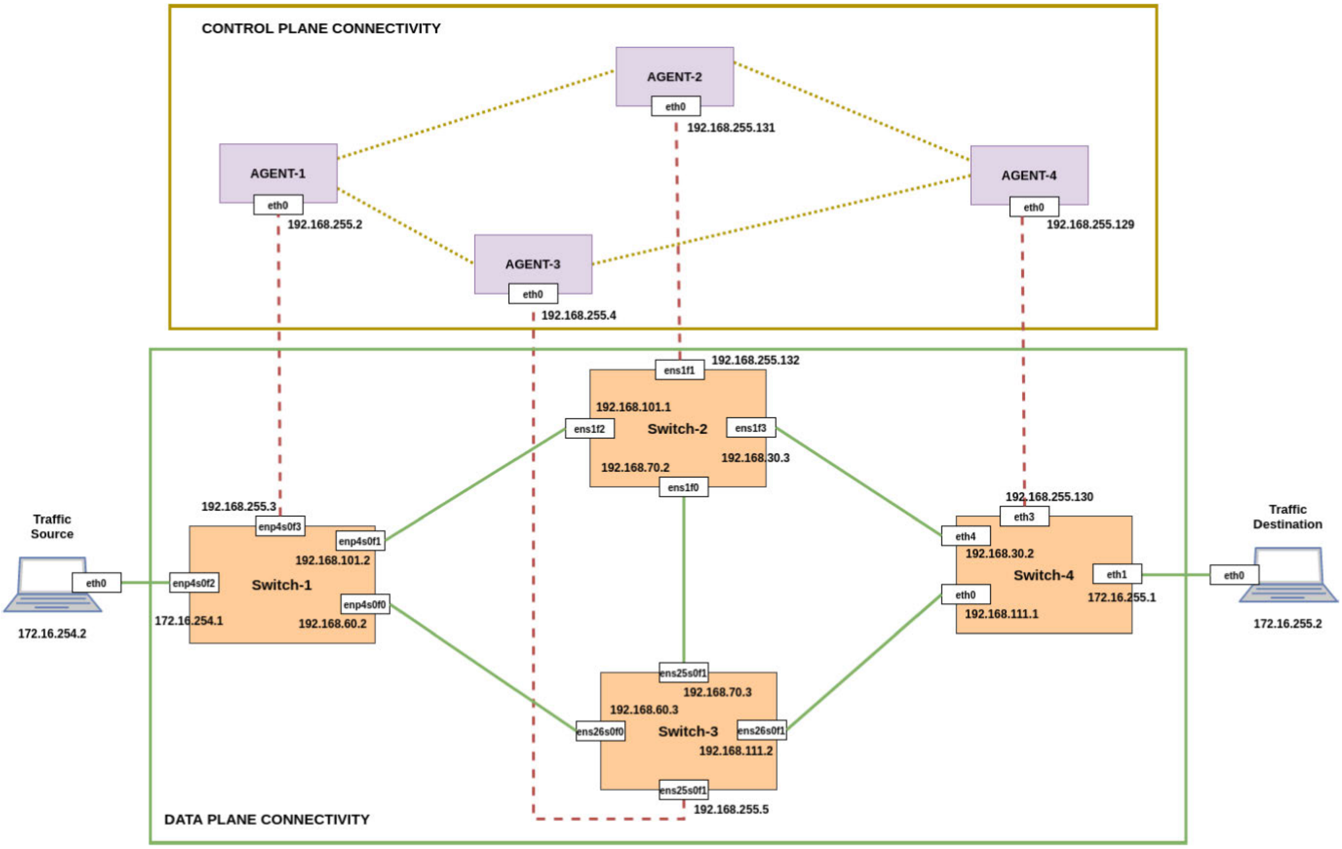


FIGURE 6. Data plane and control plane with 4 nodes.

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 TAPRIO configuration XML stored in the NETCONF Server and issues the related command. In this case, the TSN app reads the TAPRIO configuration 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:

```
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 queues
        |  +--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.

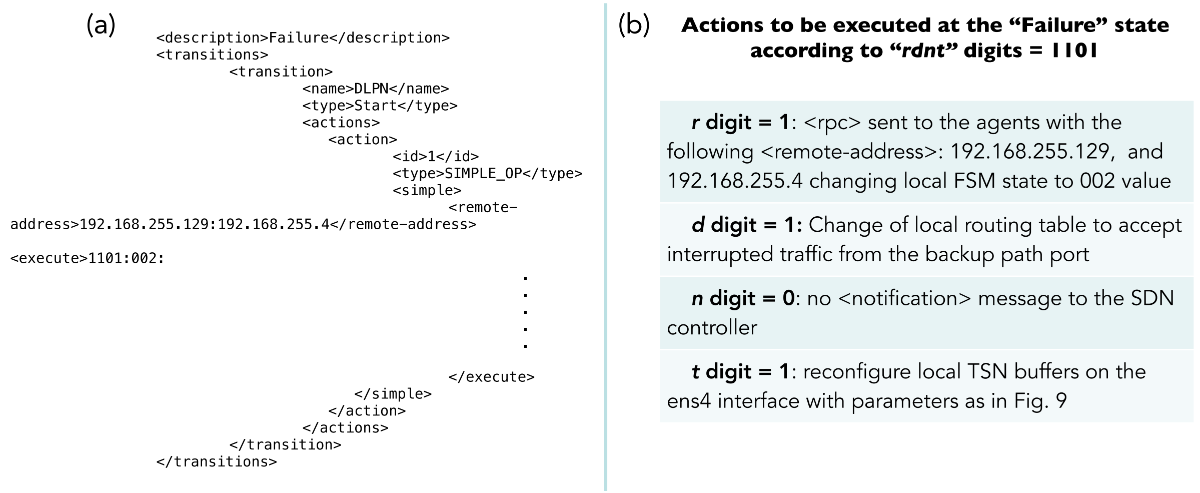
- *interface*: it represents the list of interfaces at the node, each one identified by the string *dev*
- *dev*: the name of the interface
- *parent*: class id. Default: root
- *handle*: unique handle identifier for qdisc
- *num-tc*: number of traffic classes. Max 16;
- *maps*: list to map priority to traffic classes;
- *queues*: identifies a list of queues, where each element is identified by *id*
- *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, while Fig. 8(b) 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 ':': i) 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 *rdnt* 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. 8(b). 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



(a)

```
                <description>Failure</description>
                <transitions>
                        <transition>
                                <name>DLPN</name>
                                <type>Start</type>
                                <actions>
                                   <action>
                                          <id>1</id>
                                          <type>SIMPLE_OP</type>
                                          <simple>
                                                <remote-
address>192.168.255.129:192.168.255.4</remote-address>

<execute>1101:002:
                                                        .
                                                        .
                                                        .
                                                        .
                                                        .
                                                   </execute>
                                       </simple>
                                   </action>
                                </actions>
                        </transition>
                </transitions>
```

(b) **Actions to be executed at the "Failure" state according to "*rdnt*" digits = 1101**

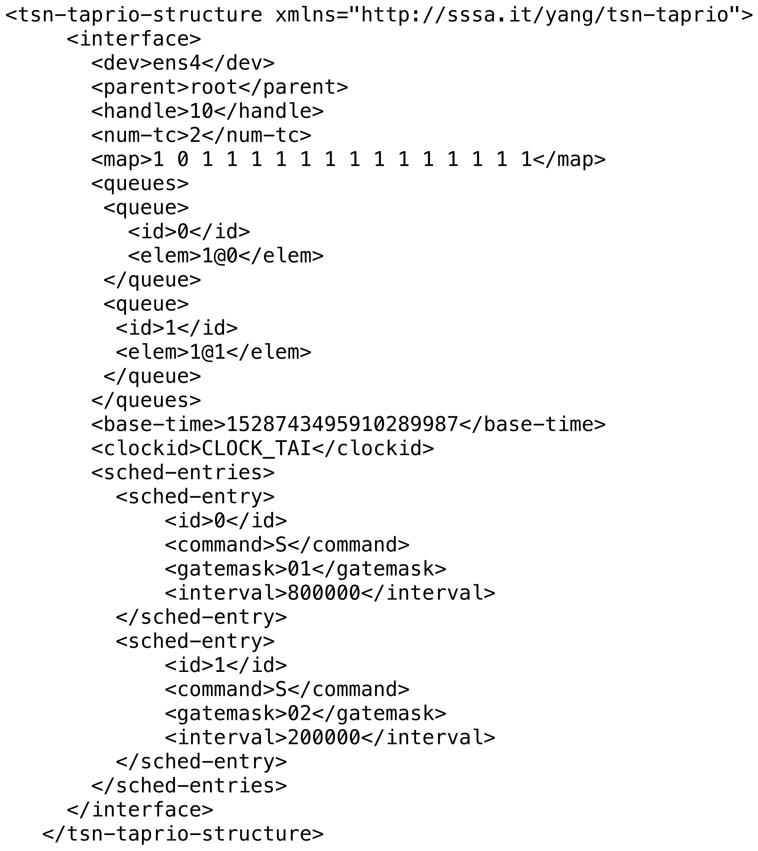
***r* digit = 1**: <rpc> sent to the agents with the following <remote-address>: 192.168.255.129, and 192.168.255.4 changing local FSM state to 002 value

***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

FIGURE 8. Installed FSM (a); Summary of the actions to be executed (b).



```
<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>S</command>
             <gatemask>01</gatemask>
             <interval>800000</interval>
         </sched-entry>
         <sched-entry>
             <id>1</id>
             <command>S</command>
             <gatemask>02</gatemask>
             <interval>200000</interval>
         </sched-entry>
       </sched-entries>
     </interface>
   </tsn-taprio-structure>
```

FIGURE 9. TAPRIO configuration XML.

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 TAPRIO configuration XML parameters, as shown in Fig. 9, with values associated to the attributes of the YANG model in Fig. 7.

D. 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 update the local routing table.

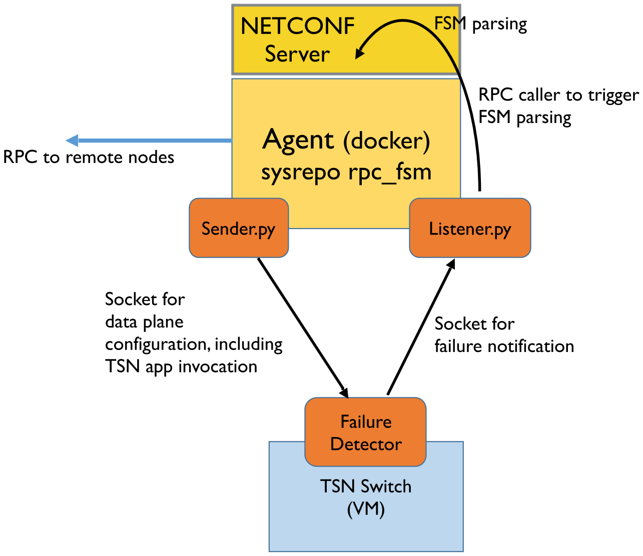


FIGURE 10. Agent implementation.

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.

TABLE 2. Characteristics of the switches.

Switch	Linux kernel	Num of Cores	RAM [GB]
Switch-1	5.8.0-48-generic	20	80
Switch-2	5.8.0-48-generic	8	8
Switch-3	5.4.0-70-generic	8	16
Switch-4	5.4.0-70-generic	8	32

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 to reduce recovery delay with respect to CR because DRTSN enables a direct communication between the node detecting the failure and the nodes involved in the recovery path, without involving a communication also with the SDN controller. Time contributions to the recovery delay in the case of DRTSN will be detailed in Sec. VI-B.
2) *Experiment 1*: 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. The behavior of DRTSN and CR is confirmed with the former reducing the recovery delay. With DRTSN, the recovery delay increases with the number of nodes. In particular, with two nodes, DRTSN requires around 520 ms, while with three and four nodes 846 ms and 823 ms, respectively. Thus, with 3 and 4 nodes, the recovery delay is comparable. This is due to the fact that <rpc> messages triggering FSM

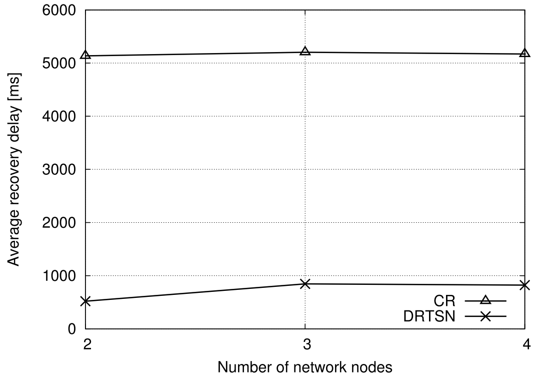


FIGURE 11. Average recovery delay [ms] vs. the number of network nodes.

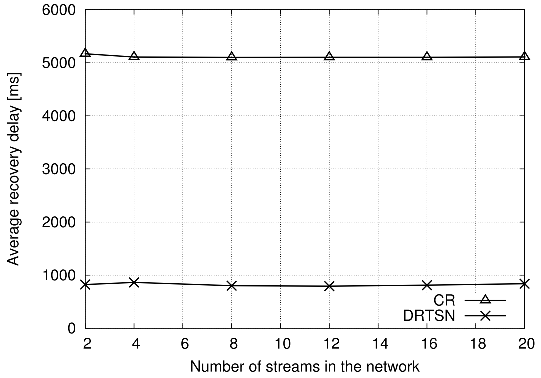


FIGURE 12. Average recovery delay [ms] vs. the number of streams present in the 4-nodes network.

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 <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 <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 streams present in the 4-nodes meshed network. Streams are generated by the Spirent and are routed randomly in the network following available paths. Then, the failure is generated and the affected streams are recovered. Fig. 12 shows the recovery delay versus the number of streams in the network. Again, the behavior of DRTSN and CR is confirmed with DRTSN reducing the recovery delay. The number of streams in the network does not impact the performance of the recovery schemes.

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 <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 <rpc> message. The <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 <rpc> and two <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 local routing table update takes around 6.32 ms.

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
Routing table update	6.32

With DRTSN, upon failure, <rpc> messages to remote nodes are exchanged. In particular, an <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 <notification> messages is sent from the node detecting the failure to the SDN controller to inform it about the failure. Then, the NETCONF <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 <rpc> generation time contribution, which includes the time to open a ssh session, a NETCONF session, and finally the time to generate the <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 <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. VI-B (requiring to open ssh and NETCONF sessions

TABLE 4. Recovery delay after DRTSN optimization.

	DRTSN w/o open NETCONF sessions	DRTSN with open NETCONF sessions
<rpc> generation time [ms]	325	7
Recovery delay [ms]	823	254

before sending <rpc> messages). The considered topology is the 4-nodes meshed one as in Fig. 6.

This optimization of DRTSN permits to reduce the <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 <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, 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/λ. Bursts of 100 frames per stream are considered. Single-link failures are randomly generated among the network links. We considered 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/λ.

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 DRTSN 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.

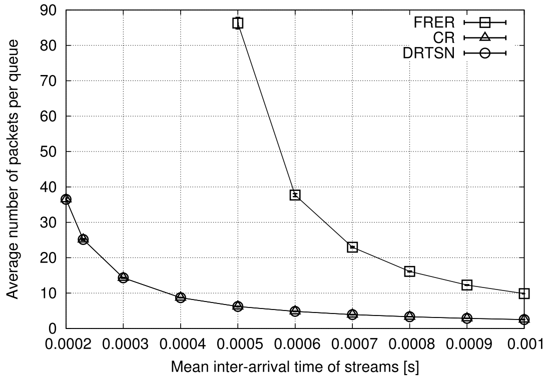


FIGURE 13. Average number of frames per buffer vs. 1/λ.

VII. CONCLUSION 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 further investigate the optimization of the proposed method. Based on the observations in the experimental testbed we have seen that a relevant time contribution to the recovery delay may be the generation of <rpc> messages. Then, future works will investigate the potentials of integrating 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.

ACKNOWLEDGMENT

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

REFERENCES

[1] *Time-Sensitive Networking (TSN) Task Group*. [Online]. Available: https://1.ieee802.org/tsn/

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

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

[4] G. Saldamli, H. Mishra, N. Ravi, R. R. Kodati, S. A. Kuntamukkala, and L. Tawalbeh, "Improving link failure recovery and congestion control in SDNs," in *Proc. 10th Int. Conf. Inf. Commun. Syst. (ICICS)*, Jun. 2019, pp. 30–35.

[5] *Frame Replicaton and Elimination for Reliability*, Standard ISO/IEC/IEEE 8802-1CB, 2019.

[6] F. von Tüllenburg and T. Pfeiffenberger, "Layer-2 failure recovery methods in critical communication networks," in *Proc. ICNS*, 2016, pp. 1–6.

[7] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "OpenFlow-based segment protection in Ethernet networks," *J. Opt. Commun. Netw.*, vol. 5, no. 9, pp. 1066–1075, 2013.

[8] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, *Network Configuration Protocol (NETCONF)*, document RFC 6241, Internet Engineering Task Force (IETF), Jun. 2011.

[9] M. Bjorklund, *YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, document IETF RFC 6020, 2010.

[10] S. Brooks and E. Uludag, "Time-sensitive networking: From theory to implementation in industrial automation," Time Sensitive Netw., Intel, Mountain View, CA, USA, Tech. Rep. WP-01279-1.0, 2018.

[11] S. Kehrer and F. Chen, "TSN configuration–focusing on network management protocols," IEEE 802.1 Working Group, Hirschmann Automat. Control, Neckartenzlingen, Germany, Tech. Rep., 2017. [Online]. Available: https://www.ieee802.org/1/files/public/docs2017/cc-kehrer-TSN-Configuration-0317-v01.pdf

[12] *TTTech Releases World's First Vendor-Independent TSN Configuration Software*. [Online]. Available: https://www.tttech.com/tttech-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*, document draft-sambo-netmod-yang-fsm-05, May 2019.

[14] M. Dallaglio, N. Sambo, F. Cugini, and P. Castoldi, ''Method for managing a telecommunication network,'' Patent U.S. 10 749 624 B2, Aug. 18, 2020.

[15] N. Sambo, ''Locally automated restoration in SDN disaggregated networks,'' *J. Opt. Commun. Netw.*, vol. 12, no. 6, pp. C23–C30, 2020.

[16] *Enhancements for Scheduled Traffic*, Standard IEEE 802.1Qbv, 2016.

[17] *IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic*, IEEE Standard 802.3br-2016, 2016.

[18] *Frame Preemption*, Standard IEEE802.1Qbu, 2015.

[19] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks—Amendment 34: Asynchronous Traffic Shaping*, Standard IEEE 802.1Qcr-2020, 2020.

[20] *Cyclic Queuing and Forwarding*, Standard P802.1Qch, 2015.

[21] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, ''Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research,'' *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 88–145, 1st Quart., 2019.

[22] W. Steiner, S. S. Craciunas, and R. S. Oliver, ''Traffic planning for time-sensitive communication,'' *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 42–47, Jun. 2018.

[23] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, ''Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks,'' in *Proc. 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2016, pp. 183–192.

[24] Z. Luxi, P. Paul, and S. S. Craciunas, ''Worst-case latency analysis for IEEE 802.1 Qbv time sensitive networks using network calculus,'' *IEEE Access*, vol. 6, pp. 41803–41815, 2018.

[25] D. Maxim and Y.-Q. Song, ''Delay analysis of AVB traffic in time-sensitive networks (TSN),'' in *Proc. 25th Int. Conf. Real-Time Netw. Syst.*, Oct. 2017, pp. 18–27.

[26] N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, ''Window-based schedule synthesis for industrial IEEE 802.1Qbv TSN networks,'' in *Proc. 16th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, Apr. 2020, pp. 1–4.

[27] S. M. Laursen, P. Pop, and W. Steiner, ''Routing optimization of AVB streams in TSN networks,'' *SIGBED Rev.*, vol. 13, no. 4, pp. 43–48, Nov. 2016.

[28] N. G. Nayak, F. Duerr, and K. Rothermel, ''Routing algorithms for IEEE802. 1Qbv networks,'' *ACM SIGBED*, vol. 15, no. 3, pp. 13–18, 2018.

[29] N. G. Nayak, F. Dürr, and K. Rothermel, ''Time-sensitive software-defined network (TSSDN) for real-time applications,'' in *Proc. 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2016, pp. 193–202.

[30] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop, ''AVB-aware routing and scheduling of time-triggered traffic for TSN,'' *IEEE Access*, vol. 6, pp. 75229–75243, 2018.

[31] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, ''ILP-based joint routing and scheduling for time-triggered networks,'' in *Proc. 25th Int. Conf. Real-Time Netw. Syst.*, Oct. 2017, pp. 8–17.

[32] M. Pahlevan, N. Tabassam, and R. Obermaisser, ''Heuristic list scheduler for time triggered traffic in time sensitive networks,'' *SIGBED Rev.*, vol. 16, no. 1, pp. 15–20, Feb. 2019.

[33] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, ''Enabling fog computing for industrial automation through time-sensitive networking (TSN),'' *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 55–61, Jun. 2018.

[34] C. Mas-Machuca, F. Musumeci, P. Vizarreta, D. Pezaros, S. Jouët, M. Tornatore, A. Hmaity, M. Liyanage, A. Gurtov, and A. Braeken, *Reliable Control and Data Planes for Softwarized Networks*. Cham, Switzerland: Springer, 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,'' U.S. Patent 10 708 179, Jul. 7, 2020.

[36] R. Amin, M. Reisslein, and N. Shah, ''Hybrid SDN networks: A survey of existing approaches,'' *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3259–3306, 4th Quart., 2018.

[37] C.-Y. Chu, K. Xi, M. Luo, and H. J. Chao, ''Congestion-aware single link failure recovery in hybrid SDN networks,'' in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 1086–1094.

[38] O. Tilmans and S. Vissicchio, ''IGP-as-a-backup for robust SDN networks,'' in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM) Workshop*, Nov. 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 Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014, doi: 10.1145/2656877.2656890.

[40] *Bridges and Bridged Networks—Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*, IEEE Standard 802.1Qcc, 2018.

[41] *ONOS—A New Carrier-Grade SDN Network Operating System Designed for High Availability, Performance, Scale-Out*. [Online]. Available: https://onosproject.org

[42] *Spirent AION*. [Online]. Available: https://www.spirent.com/products/aion

[43] J. W. J. Williams, ''Algorithm 232: Heapsort,'' *Commun. ACM*, vol. 7, no. 6, pp. 347–348, 1964.

NICOLA SAMBO received the Ph.D. degree from Scuola Superiore Sant'Anna, Pisa, Italy, in 2009. His activity is focused on networks, ranging from signal transmission to control plane, architecture, and design. He took part in several projects, such as the EU HORIZON 2020 ORCHESTRA (on monitoring and automation) and the Italian PRIN FIRST (on space division multiplexing in optical networks). He collaborates with several industrial and academic partners. He is currently an Assistant Professor at Scuola Superiore Sant'Anna. He is also associated with the FIBER CNIT National Laboratory, L'Aquila, Italy. He is an author of more than 150 publications including international journals, conference proceedings, and patents.

SILVIA FICHERA received the Laurea degree *(cum laude)* in telecommunications engineering from the University of Catania, Catania, Italy, in 2014, and the Ph.D. degree in emerging digital technologies, curriculum photonic technologies, at Scuola Superiore Sant'Anna, Pisa, Italy, in April 2019. In 2013, she was also a Trainee with the Laboratory for Communication and Applications (LCA), École Polytechnique Fédérale de Lausanne (EPFL). She is currently serving as a Research Assistant at Scuola Superiore Sant'Anna. During her Ph.D., she spent seven months with the Communication Networks Division, Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), Castelldefels, Spain. Her research interests include network control and management, software defined networking, network security, and cloud computing.

ANDREA SGAMBELLURI received the master's degree in telecommunications engineering from the University of Pisa, in 2007, and the Ph.D. degree from Scuola Superiore Sant'Anna, Pisa, Italy, in 2015. In March 2015, he won the grand prize at 2015 OFC Corning Outstanding Student Paper Competition for the paper ''First demonstration of SDN-based segment routing in multi-layer networks.'' In 2016, he was Postdoctoral Researcher at the Optical Networks Laboratory (ONLab), KTH Royal Institute of Technology. He is currently a Postdoctoral Researcher at the TeCIP Institute, Scuola Superiore Sant'Anna. His main research interests include the field of control plane techniques for both packet and optical networks, including software defined networking (SDN) protocol extensions, network reliability, industrial ethernet, switching, segment routing application, YANG/NETCONF solutions for the dynamic management, telemetry, (re)programming, and monitoring of optical devices.

GIUSEPPE FIOCCOLA was born in 1983. He received the master's degree in electronic engineering from the University of Naples Federico II, in 2008, and the postgraduate master's degree from the Polytechnic University of Turin, in 2009. He has been a Standardization Specialist and the Technology Planning and Cooperation Manager at Huawei Technologies, since 2018. Prior to Huawei Technologies, he started his career at Telecom Italia, in 2009, as a Researcher and a Network Engineer. He has more than ten years of experience in developing communication solutions and exploring new networking technologies. He is an active contributor to SDOs, such as IETF, ETSI, and MEF with relevant standard documents published. He also holds several patents in the field of IP network and performance measurement.

PIERO CASTOLDI (Senior Member, IEEE) is currently a Full Professor and the Leader of the ''Networks and Services'' research area at the TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy. He is also serving as a member of the Management Board of the Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT) and the Executive Board of the Inphotec Foundation, and he is the Director of the Erasmus Mundus Master on Photonic Integrated Circuits, Sensors and Networks (PIXNET). His research interests include telecommunications networks and system both wired and wireless, and more recently reliability, switching paradigms and control of optical networks, including application-network cooperation mechanisms, in particular for cloud networking.

KOSTAS KATSALIS is currently a Network Architect at Docomo Euro Lab, Munich, Germany. He is also an ETSI NFV and Open-RAN Standards delegate. He has served as the technical project manager for a number of projects in the area of time-sensitive-networking (TSN). He has also participated in numerous EU FP7and H2020 projects (CONTENT, COHERENT, Q4Health, and 5G-Picture). His research interests include new network designs and technologies, SDN/NFV, time-sensitive deterministic network communications, network programmability, network optimization, orchestration, and management. He serves as a Regular Reviewer in a number of conferences/journals, such as IEEE/ACM Transactions on Networking, IEEE Transactions on Network and Service Management, IEEE INFOCOM, and IEEE GLOBECOM.