SRv6-based Time-Sensitive Networks (TSN) with low-overhead rerouting

Nandha Kumar, Gagan; Katsalis, Kostas; Papadimitriou, Panagiotis; Pop, Paul; Carle, Georg

Published in:
International Journal of Network Management

Link to article, DOI:
10.1002/nem.2215

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

Received: 25 May 2022 | Revised: 24 August 2022 | Accepted: 4 October 2022

DOI: 10.1002/nem.2215

SPECIAL ISSUE PAPER

WILEY

SRv6-based Time-Sensitive Networks (TSN) with low-overhead rerouting

Gagan Nandha Kumar[1] | **Kostas Katsalis**[2] | **Panagiotis Papadimitriou**[3] | **Paul Pop**[4] | **Georg Carle**[1]

[1]Technical University of Munich, Munich, Germany

[2]DoCoMo Communications Laboratories Europe GmbH, Munich, Germany

[3]University of Macedonia, Thessaloniki, Greece

[4]Technical University of Denmark, Lyngby, Denmark

Correspondence
Gagan Nandha Kumar, Aptiv Services Deutschland GmbH, Wuppertal, Germany.
Email: gagan@tum.de

Abstract

Time-Sensitive Networks (TSN) aims at providing a solid underpinning for the support of application connectivity demands across a wide spectrum of use cases and operational environments, such as industrial automation and automotive networks. However, handling network updates in TSN entails additional challenges, stemming from the need to perform both flow rerouting and TSN schedule reconfiguration. To address this issue, we propose a software-defined network (SDN)-based approach for low-overhead TSN network updates, exploiting segment routing over IPv6 (SRv6) for path control. To this end, we introduce the concept of TSN subgraphs in order to quickly reschedule the flows traversing the problematic area and propose a TSN-aware routing heuristic to minimize the convergence time. We further describe the control plane implementation and its integration into Mininet, which empowers us to conduct a wide range of performance tests. Our evaluation results indicate that our approach yields faster recovery and reduces significantly the number of required reconfigurations upon failures, at the expense of a small SRv6 encoding/decoding overhead.

KEYWORDS

deterministic networking, IEEE TSN, network reliability, QoS, source routing, SRv6

1 | INTRODUCTION

Recent advances in the field of deterministic communications by IEEE 802.1 TSN and IETF Detnet are stimulating the adoption of Ethernet technology in several traditionally siloed environments, such as industrial automation and automotive networks. Using amendments, such as 802.1Qbv (time-aware shaper) and 802.1Qcr (asynchronous traffic shaping), performance guarantees can be achieved for multiple traffic classes, even under conditions of severe load. However, it should be noted that routing decisions employed on traffic flows with different priority strongly influences the in-class interference on the egress ports of the TSN bridges. Thus, flow priority agnostic routing techniques like

Abbreviations: TSN, Time Sensitive Networking; SRv6, Segment Routing over IPv6.

This research was conducted when the corresponding author was associated with Technical University of Munich.

shortest path leads to increased in-class interference. This in turn increases the complexity of the scheduling algorithms as more interfering traffic flows are required to be scheduled over the same link. As a result, the overall determinism of the network is dependent jointly on both the scheduling mechanism and routing policy employed.

In principle, the network convergence problem is related to routing table convergence after network updates or network failures. QoS handling is independently performed on each network device based on the priority identification in each packet. In TSNs, however, depending on the protocol in effect, after network updates or failures, path selection and scheduling are tightly coupled. Only after both routing and scheduling are decided and the appropriate configuration is applied at each device, the network is able to satisfy the appropriate KPIs (e.g., delay and jitter guarantees). Although the joint routing and scheduling problem for time-critical TSN flows has been investigated by previous studies,[1–3] poor research results exist when TSN flows are routed using a converged segment routing plane.

Source routing is used to empower the traffic originator to encode the path that each packet should traverse within its header.[4] As such, forwarding operations are performed over a nearly stateless data plane, with a minimum forwarding table size, irrespective of the number of flows and end-points in the network. Segment routing (SR), developed by IETF, reassembles the basic principles of source routing, extending it into a solution for steering traffic through a sequence of *segments*, along the network path.[5,6] In this respect, a segment list is inserted into each packet (either by the packet source or by an intermediate network node), prescribing certain processing and/or forwarding actions that should take place at each segment. Two schemes currently exist for realizing segment routing, that is, SR over IPv6 (SRv6) and SR over MPLS.

In our previous work,[7] we developed a framework for coupling IEEE TSN technology with source routing in order to manage explicit TSN paths and cope with the relevant joint TSN routing and scheduling problem. In this work, we extend our investigation and study the integration of SRv6 with IEEE TSN, as well as the relevant control and data plane aspects. Similarly to Kumar et al,[7] in order to minimize the time required for rerouting and rescheduling, we seek to isolate the area where the problem has occurred by finding a minimum connected subgraph, on which the scheduler can be invoked. With this proposed approach, we locally solve the flow rerouting and scheduling problems for the affected part of the network. As opposed to Kumar et al.,[7] TSN flow rerouting is now performed using SRv6.

Our contributions are summarized as follows:

- We propose and evaluate a novel SDN-based approach for coupling segment routing IPv6 with IEEE TSN.
- We showcase how segment routing can be employed for explicit path control of TSN flows.
- We incorporate the concept of network subgraphs to alleviate the problem of TSN scheduling complexity.
- We describe how both segment routing and TSN time-aware priority shaper (TAPRIO) queuing can be integrated into a Mininet environment.
- We conduct an evaluation study of both control- and data-plane performance.

In principle, segment routing relies on IGP protocols (IS-IS, OSPF) to distribute segment identifiers information (SIDs), whereas the segment routing control plane may also consider centralized operations regarding the computation of paths, for example, exploiting a centralized path computation element (PCE). In this work, we follow a similar approach to PCE. More precisely, we propose an enhanced path computation function and we investigate how the integration of a fully centralized TSN control plane with segment routing can be realized. SDN assisted operations for Segment routing are also discussed in Ventre et al.[8]

For enhanced reliability of TSN flows, frame replication and elimination according to 802.1CB can be used. On one hand, frame replication may increase the probability of successful reception; on the other hand it may lead to waste of bandwidth since duplicated flows are traversing the network. Depending on the application domain, this bandwidth waste may not be acceptable. Furthermore, in modern SDN-based networks, the global knowledge of the network is taken advantage by the centralized network controller in order to develop network recovery techniques with minimal recovery time. In the reactive approach, the centralized network controller is mainly responsible for the failure handling process. In case of a failure, the controller is informed of the failure event either by polling or by event-based approach. The controller then decides for alternative paths and applies the relevant configuration to the network devices. Using a proactive approach, the link recovery process is locally handled a priori before the failure event with a minimum intervention by the controller. The controller configures the forwarding table of the switches with both main and backup paths. In the event of a failure, the switches automatically re-route the packets according to the forwarding rules related to the backup path. See Sambo et al[9] for our previous work on proactive SDN-based TSN failure handling.

The motivation for this work stems from the importance of segment routing, as a new routing strategy applicable in multiple different application domains, where IEEE TSN is also considered. For example, in the mobile network recent

studies are also investigating the role of segment routing supporting Transport network functionality, where at the same time IEEE TSN is already integrated as part of 3GP Release 16 and beyond. No solid framework or solution, however, exists when considering control and dataplane aspects for the converged network in the light of failure events.

The remainder of the paper is organized as follows. Section 2 discusses related work, Section 3 presents the system model and the problem statement, whereas in Section 4, we elaborate on the proposed solution. In Section 5, we present the evaluation environment and the evaluation results. Finally, Section 6 highlights our conclusions and outlines directions for future work.

2 | RELATED WORK

Legacy techniques, such as class-based weighted fair queueing (CBWFQ) and deficit round robin,[10] can be applied to preserve accurate throughput guarantees. However, providing delay/jitter guarantees in vanilla ethernet is not possible, if such techniques or strict priority scheduling are used together with 802.1p-based prioritization, because of the strong dependency on the stochastic nature of traffic. IEEE TSN offers a rich set of tools and mechanisms to enable accurate stream prioritization in a way to also provide delay and jitter guarantees. TSN is gaining momentum as the future key technology to enable deterministic performance for a number of challenging applications, such as in industrial networks (IEC/IEEE 60802 profile), automotive (P802.1DG profile) and service provider networks (P802.1DF).

In the following, we provide background information and discuss related work on TSN, DetNet, source routing, and segment routing.

TSN technology: Techniques used to provide delay guarantees are scheduled traffic (IEEE 802.1Qbv), frame preemption (IEEE 802.3br, IEEE 802.1Qbu), asynchronous traffic shaping (802.1Qcr), and cyclic queuing and forwarding (802.1Qch). These standards define how frames belonging to a particular traffic class or having a particular priority are handled by TSN-enabled bridges. In this work, we focus on IEEE 802.1Qbv which introduces a transmission gate operation for each traffic class queue. The transmission gates are in an *open* or *closed* state and are 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. In terms of the TSN control plane, three models have been proposed in the 802.1Qcc amendment. In the fully centralized case, which is the one we consider in this study, the flow requirements are conveyed from a centralized user configurator (CUC) to a centralized network configurator (CNC), using a user network information (UNI) described in 802.1Qdj.

TSN path control: In 802.1Qcc, several path control protocols are supported, such as STP, MSTP, RSTP, and SPB. An alternative approach to 802.1Qcc is 802.1Qca used for explicit path selection (explicit trees) and bandwidth reservation. 802.1Qca relies on *IS-IS* routing protocol to carry control information and a Path Computation Element (PCE) to identify the optimal path. However, the propagation of the control information is passed with IS-IS in a distributed manner, which leads to slow convergence upon failures.

802.1Qbv scheduling: For a broad view on TSN, we refer interested readers to Nasrallah et al.[11] In previous studies,[12,13] 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 previous works.[14,15] More recently, window-based schedule synthesis[16] has been proposed for industrial IEEE 802.1Qbv TSN Networks with unscheduled end-systems. In Laursen et al,[17] the problem of computing 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. Nayak et al[18] explore how the routing of time-triggered flows affects their schedulability and propose integer linear programming (ILP)-based routing algorithms. Various ILP formulations for the combined routing and scheduling time-triggered traffic problem, while following the SDN-based paradigm are presented in Nayak et al.[19] Gavriluţ et al[1] propose a joint routing and scheduling approach for both time-triggered (TT) and audio-video-bridging (AVB) traffic. An ILP scheduling and routing formulation to improve the TT traffic schedulability is proposed in Schweissguth et al.[2] A list scheduling-based heuristic scheduling and routing algorithm is proposed in Pahlevan et al.[3] The work in Pop et al[20] has shown how to reconfigure the GCLs at runtime, for example, as a reaction to network changes, potentially due to failures.

Network recovery: Routing-driven topology synthesis is addressed in Atallah et al[21] for fault-tolerant TSN networks but nonscheduled traffic. Reliable control and data planes for SDN have been investigated in Mas-Machuca et al.[22] A method of handling data packets through a conditional state transition table for implementing at least one finite state

machine is proposed by Bianchi et al.[23] Fast fail-over has been proposed in von Tüllenburg and Pfeiffenberger[24] for OpenFlow environments. A similar approach has been employed by Saldamli et al,[25] also managing congested links. In Sgambelluri et al,[26] a segment-based recovery has been proposed for OpenFlow. With these approaches, a node detecting the failure steers traffic to a backup path. A different approach is taken by IEEE 802.1CB, which specifies the procedures for frame replication and elimination (FRER).

Source routing: Source routing enables the encoding of the forwarding path into the packet header.[4,27,28] More precisely, upon path computation, the path is encapsulated into the packet as a sequence of labels, which may correspond to or match the sequence of output ports that a packet needs to traverse in order to reach its destination. This obviates the need for maintaining a forwarding state in switches for all the destinations in the network.[28,29] A port switching scheme for source routing is presented in Hari et al.[30] In source routing various header fields can be utilized for path encoding, such as source/destination MAC and IP(v6) addresses,[27,28] or MPLS.[31] In our previous work,[7,32] we investigate the coupling of source routing with TSN on the way explicit paths can be managed for time-critical flows.

Segment Routing (SR): Based on the source routing paradigm, SR comprises an architecture for steering traffic through a sequence of segments, that is, way-points along the network path. A segment list is inserted into each packet (either by the packet source or by an intermediate network node), prescribing certain processing and/or forwarding actions that should take place in each segment. Segment routing has been realized on top of MPLS[33] and IPv6.[34,35] Segment routing has been considered as a promising approach to traffic engineering, monitoring, and failure recovery.[6] In terms of traffic engineering, Bhatia et al[36] deal with the problem of determining the optimal parameters for segment routing in offline and online cases. The paper focuses on two-segment routing, by arguing that one intermediate node strikes a good balance between shortest path routing and the extra transmission and processing overhead required for multiple segments. In Aubry et al.,[37] SCMon leverages on segment routing for the continuous monitoring of the data-plane and the ability of link failure and overload detection. SCMon pays particular attention to the computation of probing cycles to reduce the monitoring overhead. Aubry et al[38] stress on the need for the construction of disjoint paths, which remain as such (i.e., disjoint) even after failures, with no external intervention. This can comprise a value-added service for certain ISP clients. The computation of disjoint paths is performed using segment routing.

DetNet, Segment Routing and TSN: IETF deterministic networking (DetNet) data plane framework is defined in RFC 8938, whereas IP over IEEE 802.1 TSN is described within RFC 9023. In Chen et al,[39] the SR identifier (SID) is used to specify transmission time (cycles) of a packet. The segment ID (SID) is used to carry and specify the *sending time* of a packet, and some mechanisms can be used to ensure that the packet will be transmitted in that specified period of sending time, which is called time-aware segment routing (TA-SR). Wang et al,[40] in the light of Detnet activities, propose a solution where the TSN layer 2 header and application payload carried by the TSN network are encapsulated in the IPv6 payload field. The underlying network between the edge nodes needs to support IPv6 according to RFC8200, and can transform TSN data flow into SRv6 service. IPv6 options for DetNet are discussed in a previous work[41] with a focus on IEEE 802.1CB (frame replication and elimination for reliability) and DetNet's strict path option.

To summarize on the key aspects of our work, compared with the aforementioned previous work, to the best of our knowledge, there has been no research for explicit path routing with minimizing the TSN convergence time. In this regard, we propose and design a near stateless TSN data plane by moving the flow-specific routing information from the network nodes to the packets, thereby achieving explicit path routing in TSN with minimal overhead by reducing the number of interactions between the network devices. In this respect, we develop a reactive SDN-based approach used to minimize the failure recovery time for TSNs using the explicit path routing control plane architecture coupled with the concept of TSN subgraphs to simplify the scheduling decision making. In more detail, to minimize rerouting and rescheduling times, we seek to isolate the area where the problem has occurred by finding a minimum connected subgraph on which the scheduler can be invoked without modifying the existing schedules.

Similar to our previous work,[7] in order to minimize rerouting and rescheduling times, we find a minimum connected subgraph on which the scheduler can be invoked. In this study, however, we extend our previous work in multiple directions. More specifically, instead of source routing, we investigate segment routing technology operating on IPv6 (SRv6) on the way explicit paths are established. We propose and evaluate a hybrid SDN-based control plane model, which relies on Netconf-based configuration of SRv6-based aspects, a centralized TSN control plane, and distributed OSPF control plane operations for the distribution of the SRv6 identifiers. In our previous work,[7] we exploited a fully centralized SDN control plane model used to control TSN and source routing operations.

3 | SYSTEM MODEL AND PROBLEM STATEMENT

TSN supports the convergence of multiple traffic types, such as critical real-time time-triggered traffic, audio-video bridging traffic with guaranteed bandwidth and constrained delay requirements, and best effort traffic with no timing guarantees. Based on their timing requirements, these traffic types are identified by different priorities according to 802.1Q, also referred as flow priorities throughout the rest of this work.

The network is modeled as a network graph G denoted by $G=\{V,E\}$, where $V=\{v_i\}$ indicates the set of vertices (end-points plus network bridges) and $E=\{e_{i,j}\}$ corresponds to the set of directional edges, (source i to destination j). Let $F=\{f_i^p\}$ denote the set of all flows in the network, where $p\in\{0,1,...7\}$ expresses the flow priority. Similar to Gavriluț et al,[1] we model the time-triggered traffic as follows. Every flow is characterized by T period in microseconds, k frames sent every T microseconds, D maximum end-to-end latency in microseconds, $s\in E$ source node, and $d\in E$ destination node. Within the period T, the last frame in the k sequence must be fully received within the deadline. For the GCL compilation, the network cycle equals the least common multiple (LCM) of all the flow periods. Regarding the queuing structure at each egress port of each switch, we consider that eight traffic classes are mapped to a corresponding queue based on the flow priority (802.1Qbv). We define a binary variable $x_{e_{i,j}}^{f_k^p}$ to indicate whether edge $e_{i,j}$ belongs to the path of flow k. If flow f_k^p is routed via the edge $e_{i,j}$ then $x_{e_{i,j}}^{f_k^p}=1$ or else equals zero.

In the case of a failure event at time t, we define $F_{err}(t)\subseteq F$ as the set of all flows that are directly affected by the failure. For example, the network depicted in Figure 1 is traversed by flows $F=\{f_1^7,f_2^7,f_3^5\}$. At some point t, assuming that node 6 fails, then the directly affected flows are $F_{err}(t)=\{f_2^7,f_3^5\}$.

We define *TSN network convergence time*, as the time required to have all flows in the network scheduled and all routing paths configured in a way where all talker–listener pairs can exchange messages according to the TSN scheduling policy in effect. Our technical objective in this work is related to minimizing TSN network convergence time after network updates or network failures in case TSN flows are routed following SRv6.

Broadly speaking, network recovery is classified either as *proactive* or *reactive*. With the proactive methods, alternative paths and actions are precomputed before the failure, whereas in reactive methods either an SDN controller is notified and takes the appropriate restoration action (i.e., installs new forwarding entries) or an STP-based distributed protocol provides the new loop-free network topology. While each approach has its pros and cons in 802.1Qbv based TSN networks, additional challenges are entailed, due to the fact that the policy in effect regarding traffic class scheduling is heavily dependent on the routing strategy and the selection of the egress ports. In STP-, RSTP-, and MSTP-based networks, we cannot derive the optimal scheduling policy if the system has not converged. In proactive methods, anticipating all possible failures and providing optimal scheduling, routing decision making could be an option for small topologies; however, at a large scale, this is extremely challenging. In this study, we focus on the fully centralized case, and in addition to minimizing TSN network convergence time, we also investigate the overall communication overhead between the SDN controller and the network devices.

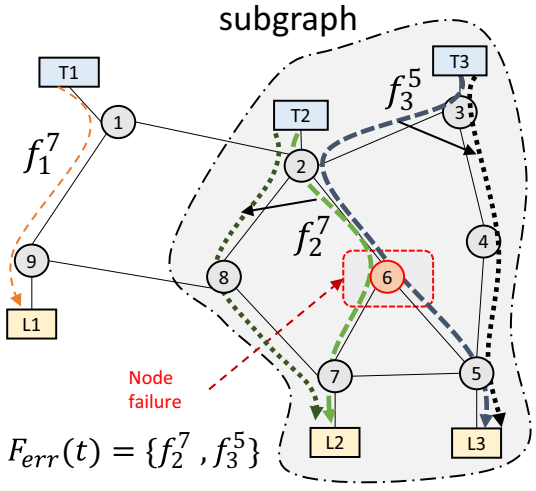

FIGURE 1 Network subgraph example with flow rerouting after failure

4 | PROPOSED SOLUTION AND TECHNICAL APPROACH

In our design, the solution is described considering the fully centralized TSN control plane, as defined by 802.1Qcc. However, the same principles could be applied when, for example, the centralized network/distributed user model are used. The system design is depicted in Figure 2. For simplicity, we consider that the TSN centralized network controller (CNC) is integrated into a centralized SDN controller with a network-wide view, which also supports legacy SDN functionalities, such as topology information handling, network discovery, statistics, and monitoring. In our approach, TSN-related aspects are considered part of the integrated SDN control plane. In our study, we do not devise a joint optimization TSN routing and scheduling problem. Instead, the routing problem is solved by a path control function hosted in the controller. After routing has been computed, the TSN scheduling problem is solved by CNC. In further detail:

- **Centralized network controller (CNC):** The CNC is responsible for the computation of 802.1Qbv TSN schedules and configuration of TSN features for all involved network devices (e.g., following 802.1Qcw). A CNC has complete knowledge of the network topology and all the streams flowing through the network, based on its interaction with the SDN and the CUC, respectively.
- **Centralized user configurator (CUC):** CUC interacts with CNC to pass TSN stream related requirements. CUC is responsible for discovery of end-stations, retrieval of end-station capabilities, and configuration of end-points. Furthermore, the CUC passes to CNC the flow specification from the end-devices for allocating network resources. The CUC forwards the flow's specification to the CNC via the user network interface (UNI) (e.g., 802.1Qdj).
- **Path control function (PCF):** Inside PCF, we consider the operation of a *TSN Network subgraphs Function*, which finds the optimal subgraphs after a network update or a failure event and a *Flow Priority Aware Routing (FPAR)* function, which executes a heuristic responsible for determining the optimal routing strategy. After the optimal strategy has been identified, PCF signals CNC to compile the GCLs according to 802.1Qdj. For the actual path establishment, we rely on SRv6. In Section 4.2.3, we describe both control-plane and data-plane aspects, when integrating SRv6 with IEEE TSN 802.1Qdj. Another option for path establishment is the use of source routing, which was investigated in Kumar et al.[7] Note that, as defined, the *Path Control function (PCF)* reassembles properties of legacy path computation function (PCF) belonging to path computation element (PCE), which in turn resides within a PCE controller. In our approach, all this functionality is considered as a function of the SDN controller.

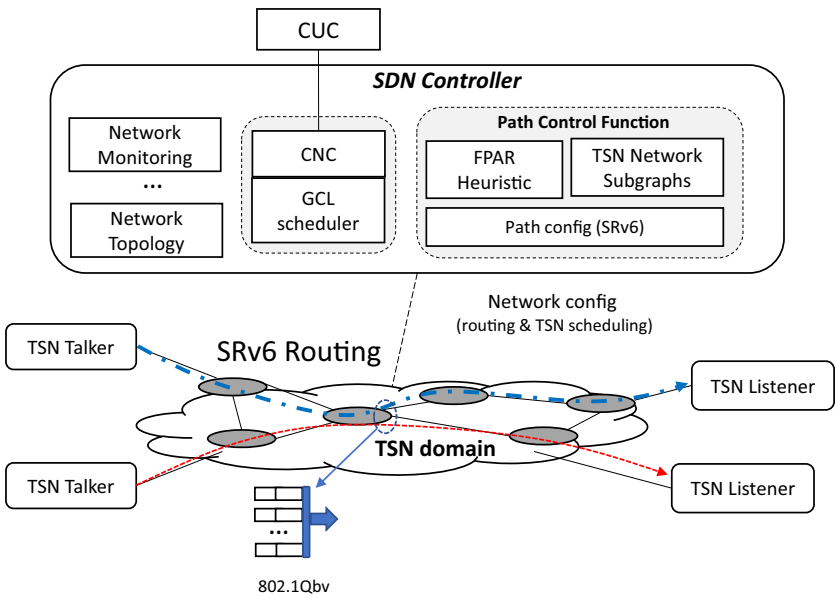

FIGURE 2 Network subgraph example with flow rerouting after failure

4.1 | Complexity minimization through network subgraphs

In order to minimize rerouting and rescheduling times, we seek to isolate the area where the update or problem has occurred. The scheme was originally presented in Kumar et al,[7] and we briefly summarize it to ease reading. The goal is to reroute and reschedule all the flows in set F_{err} without modifying the existing schedules for ports through which unaffected flows are traversing. In more detail, for the set F_{err} with flows traversing the problematic area, we aim at finding a minimum connected subgraph that could be used to reroute and reschedule these flows. For the creation of the subgraph, we devise a new variant of load balancing scheme referred to as *Flow Priority Aware Routing (FPAR)* heuristic that considers both spatial locality and flow priorities. In more detail, we derive the subgraph as a union of the links obtained by the computation of weighted k-shortest path algorithm.

Policy definition: For any given link $e_{i,j}$, we define the weight of the path between source s to destination d, which characterizes how loaded the link is

$$W^{p}_{e_{s,d}} = 1 + \beta \sum_{\forall f^{q}_{v} \in F, q \geq p} x^{f^{q}_{v}}_{e_{i,j}} + (1-\beta) \sum_{\forall f_{v} \in F} x^{f^{p}_{v}}_{e_{i,j}} \tag{1}$$

where $p \in \{0, 1, \ldots 7\}$ denotes the priority set. This weight is used when searching the least loaded links with the weighted shortest path algorithm. The second term of the equation describes how loaded the link is with traffic of priority equal or higher than p, while the third term considers all load irrespective of priority. The value of β is chosen such that the probability of a flow being allocated a routing path that already has the same priority or higher priority traffic flowing through it is minimized. For our experiments, we choose an arbitrary value of $\beta = 0.8$. This algorithm ensures that the traffic flows are allocated in those paths which offer minimal interference for the flow of time-critical traffic. This approach of routing helps reduce the scheduling complexity and increases the flexibility of finding an optimal solution by reducing the number of flows to be scheduled on a given link at any point in time.

Concrete steps: The various steps involved in the network recovery process are summarized as follows:

1. The network is continuously monitored for node and link failures.
2. In case of a failure, we mark the corresponding node/links with the error and identify the flows which are directly affected by the failure.
3. We define F_{err} as the set of all the flows that are directly affected by the failure. In the next step, our aim is to find a subgraph using routing paths calculated using FPAR to directly yield a non-overlapping path with existing unaffected flows.
4. We rank flows in F_{err} by priority and run weighted k-shortest path (according to Eq. 5.1), in order to find paths and solve the routing problem between the talker and listener. Two flows are marked as overlapping, if there exists at least one trunk where these flows are statistically multiplexed. To investigate if such a nonoverlapping path exists, we process the flows in F_{err} one-by-one starting by setting $k = 0$, the following cases of overlapping are possible:
   - **Case 1:** No overlapping exists between any flows in F: The routing path computed for each of the flows is committed without the necessity to invoke the scheduler as no interference exists between the flows (they get full capacity and there is no resource sharing).
   - **Case 2:** Overlapping exists between F_{err} but not with flows in $F - F_{err}$: We now derive the subgraph of the affected area as a union of the routing paths identified. The union may derive a connected or non-connected subgraph (example in Figure 1). Over the derived subgraph, we then perform scheduling per-flow wise using the scheduling heuristic described in Gavriluţ et al.[1] For each flow, the GCL calculation derives the minimum window needed to minimize queuing time and then is translated into a GCL configuration. This approach will also facilitate problem-solving for future failures, because it seeks to minimize resource utilization. If a solution exists for all flows in the set, we admit the schedule.
   - **Case 3:** Overlapping exists between flows F_{err} and $F - F_{err}$: In this case, We derive the subgraph over all the overlapping set of flows similar to case 2. However, the computation of schedules is now performed in an incremental manner preserving the existing schedules of the unaffected flows $F - F_{err}$ using the scheduler implementation from Gavriluţ et al.[1] Thereby, minimizing the impact of GCL reconfiguration on the other unaffected flows.
5. If for $k = 0$, there is no feasible solution exists, we continue the investigation until we reach some arbitrarily chosen value $\hat{k}_{max}$. In each case, commit operations are made per set, not per flow. If we reach $\hat{k}_{max}$ for all flows F_{err} and

there is no feasible solution exists, we then apply the policy defined in Gavriluț et al[1] over the entire network topology for all the flows in the network. variation.

We highlight that our reactive failure approach can be even more efficient when assisted by the policy in effect during the initialization phase. This is the case where the policy is operating in such a way that will make it easier to reroute/reschedule traffic in the case of failures during runtime. For example, our weighted *k*-shortest path-based routing, other routing policies tolerant to network failures policies can be employed in order to avoid overutilizing specific links and creating single points of failure. Due to space limitations, this aspect will be investigated in detail in future work.

4.2 | SRv6 and IEEE TSN 802.1Qbv integration

In our previous work[32] and Kumar et al,[7] we showcased the coupling of IEEE TSN with source routing techniques. Following the source routing principles, for each flow we encoded in the access switch where the talker is connected, the sequence of the ports that need to be traversed in order to reach the destination node. In Kumar et al,[32] a raw encoding of the path information is inserted into the MAC header, whereas in Kumar et al,[7] we extended this encapsulation following 802.1ah and developed a source routing encoding over provider backbone bridge (PBB) ethernet tunnels in order to support TSN-based flows path establishment. In both cases, at each hop every TSN switch only has to decode this information from the header of each receiving frame and perform the forwarding decision accordingly, obviating the need for L2 forwarding.

As in this work, we plan to harness the benefits of segment routing, in the following we discuss the integration of SRv6 with TSN based on IEEE 802.1Qbv.

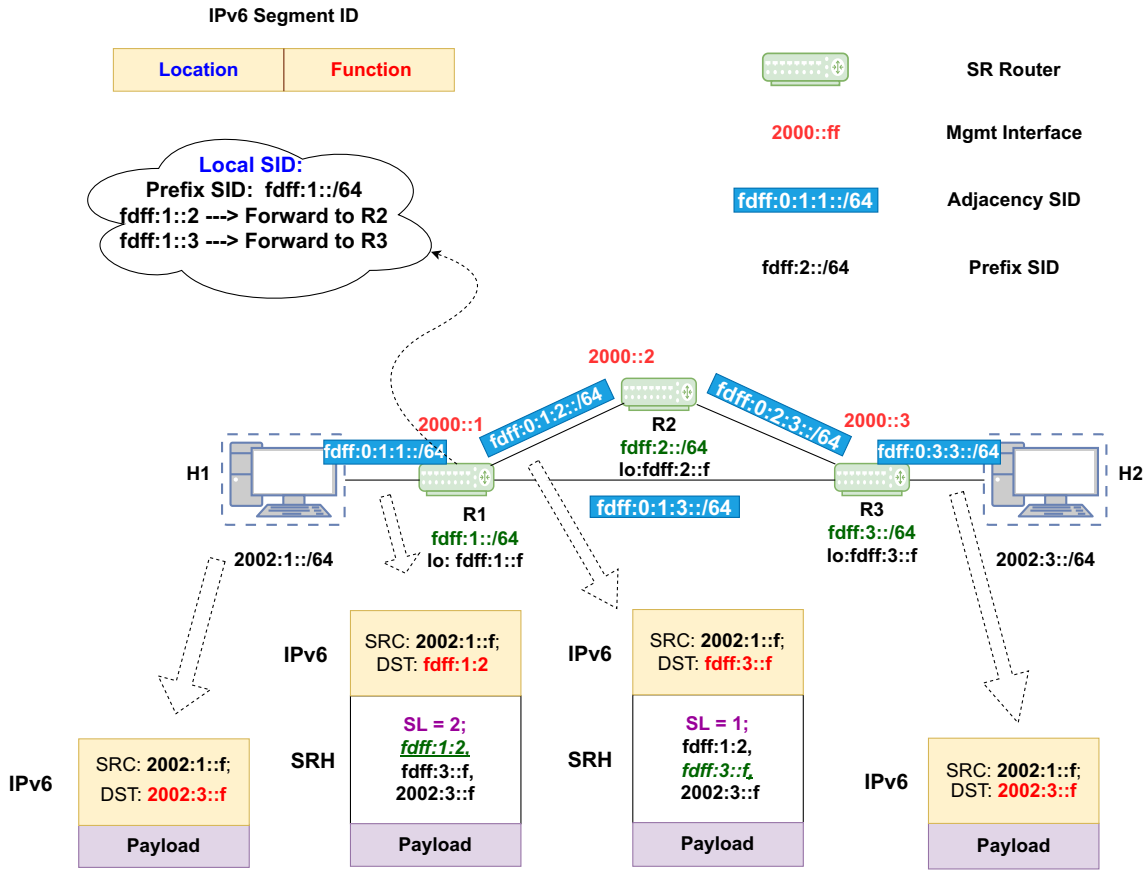

FIGURE 3 SRv6 example

4.2.1 | SRv6 basic principles of operation

In SRv6, a segment routing header (SRH) is used to insert a SR policy as an ordered list of IPv6 addresses, where each IPv6 address refers to a segment or an instruction to be processed. The current segment to be processed by a node is indicated by the *Segments Left* field, which is decremented by a SR-enabled router after processing the corresponding Segment ID.

Figure 3 illustrates the operation of SRv6 forwarding for a packet sent by H1 to H2. In an SRv6 architecture, the network is divided into multiple segments, where each segment is associated with a *Location* and a *Function* to be carried out by any associated router. In SRH, the first 64 bits of the IPv6 address indicate the location or forwarding information, while the remaining 64 bits indicate the functionality to be carried out by the network devices. In Figure 3, each node in the network consists of the prefix SID which indicates the location of the node in the network. All nodes are assumed to be reachable by every other node in the network with the help of interior gteway protocol (IGP) or by static routes. The prefix SID forms the prefix for any segment associated with a given node. For instance, consider Router R1 in this figure. The node is associated with two local segment IDs, that is, fdff:1::2 and fdff:1::3, which share the same prefix as its prefix SID, while having different suffix parts to map to different functionalities. A function could be a simple forwarding operation or even a complex operation to be performed such as network telemetry or SFC. The egress port of the host machine is responsible for encapsulating the IPv6 packet with the SRH consisting of the list of segments to be traversed. For instance, to route packets from H1 to H2 via routers R2 and R3, the egress port adds the segment list fdff:1::2, fdff:3::f to the SRH header and replaces the destination address with the first segment to be reached. The routing between the segments takes place using normal IPv6 forwarding. On reaching node R1, it performs the required functionality as indicated by the segment. In this case, the required functionality is to forward the node to R2. Hence, R1 decrements the "segments left" field by 1 and replaces the IPv6 destination address with the next segment to be traversed in the list. The packet is subsequently routed to R3 via normal IPv6 forwarding. When R3 has been reached, which is the last segment in the list, the SRH is popped out from the packet, which is eventually delivered to the destination.

4.2.2 | Proposed solution

In the following, we describe a Linux-based TSN SRv6 architecture. On the node level, the solution is based on Linux SRv6 implementations and TAPRIO Qdisc. The control plane architecture follows a hybrid SDN approach that relies on the co-existence of a distributed IP routing protocol and a centralized TSN architecture. Distributed IP routing protocols and, in particular OSPF, are responsible for the distribution of prefix SIDs across the network nodes. In our solution, we rely on OSPFv3. Figure 4 depicts the proposed architecture. Besides the generic functionalities of the basic components (i.e., CNC, CUC, and FPAR heuristic) as described in the previous section, the following operations are supported:

Control plane:

- CNC configures TSN functionality in the router (e.g., the GCLs) via a NETCONF plugin residing in the SDN controller, based on the 802.1Qcw YANG data model.
- **Topology management:** Each OSPF-enabled network device has the same view of network topology as all other nodes belonging to the same network domain. The initial topology is fetched by the SRv6-TSN application by reading the OSPF database from any one of the routers in the network. The topology information is stored locally on the controller as a network graph using the NetworkX[42] python library. In the event of a network update or failure, in order to avoid the OSPF large convergence time, any topology change such as link or node failure is tracked by the controller in the following manner. All the nodes continuously keep track of the network status of all ports with the help of a network monitor daemon. In the event of port state changes, the corresponding router notifies the SRv6-TSN control application via the management interface. The SRv6-TSN control application updates accordingly its locally stored topology information and initiates the recovery process.
- **config SRv6:** The SDN controller is responsible for the configuration of local segment IDs in the intermediate routers. In addition, for each TSN talker, it encapsulates the segment routing Path to be followed in its access device. Note that in our approach, we consider that all the network devices are 802.1Qbv TSN and SRv6 capable.

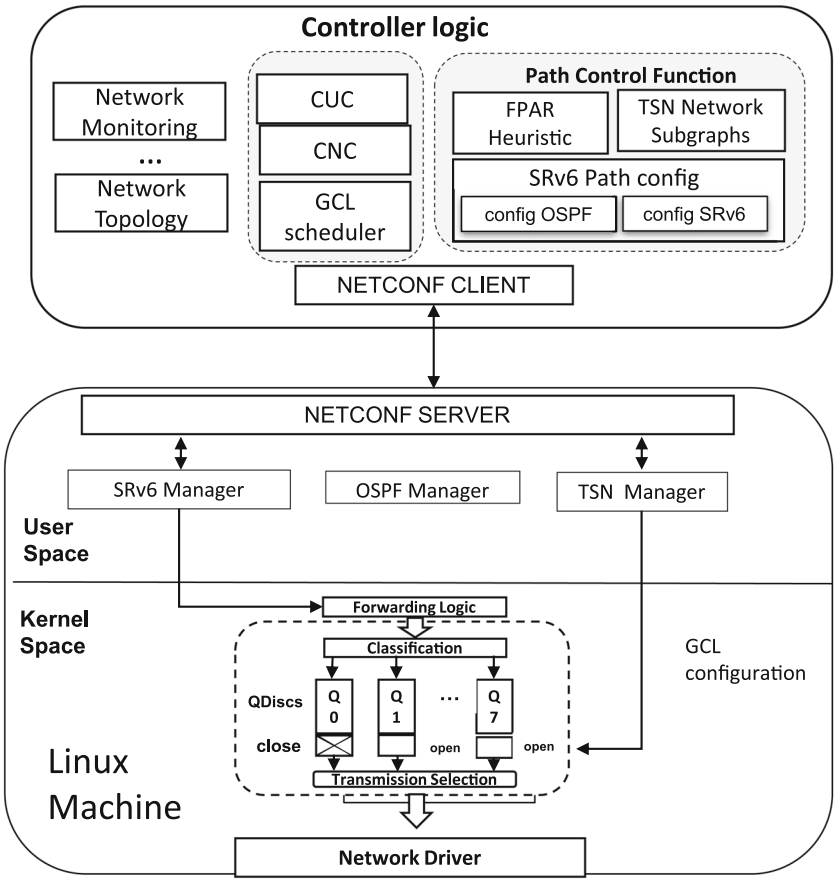

FIGURE 4 Generic controller architecture for the TSN SRv6 enabled network nodes. The same network node architecture is used for both the endpoints (TSN Talkers/Listeners) and intermediate TSN devices. SRv6 tagging operation is only performed at the TSN Talker nodes by the controller. From TSN perspective, the controller is providing both CNC and CUC functionality, including also generic SDN related operations, such as topology management

Path Control Function also includes a customized Python-based application, which serves as the link between the distributed IP Routing protocol plane and the centralized TSN control plane. It is responsible for transforming the network topology into a list of segments and assigning the node and the adjacency segment IDs. For each flow, after the routing path is selected according to FPAR/Subgraphs operation, each flow is transformed to a list of segments to be traversed. The SRv6 Path config operation considers that the routing paths are transformed to a sequence of edges (adjacency SIDs) instead of a sequence of nodes (node SID/Prefix SID) to be traversed. This is because, during topology update events like link failure, if the SR policy is defined as a sequence of nodes, the OSPF convergence takes precedence in finding the alternative paths, leading to packets flowing through the network without proper GCL configurations. However, in case the SR policy is defined as a sequence of edges to be traversed, the OSPF convergence process will not be able to recover the stream as the corresponding adjacency SID becomes invalid after the topology update. Thereby, the SDN controller can have complete control of the topology update process.

SRv6 and 802.1Qbv in a Linux node:

- **SRv6 Manager:** We consider a Linux-based SRv6-enabled device according to previous works.[8,43] The SRv6 manager is responsible for parsing SRv6 configuration information sent to the NETCONF server. Configuration information encompasses (i) path encapsulation in SRH according to YANG-based SRv6 models, translated into a suitable set of actions that tune the forwarding logic at the access devices (*srv6-explicit-path* YANG model) and (ii) mapping of segment IDs to functions (*srv6-local-sid* YANG model). In this work, both models are based on the ones presented in Ventre et al.[8] Our SRv6 Manager implementation internally relies on the *iproute* utility in order to apply the SIDs allocation (in all network devices), as well as the routing policy configuration (at the access devices).

- **TSN manager:** It is responsible for parsing GCL configuration sent over the NETCONF interface, based on 802.1Qcw aligned YANG models, and for applying the corresponding set of actions that affect the queuing disc layer of the Linux kernel (TAPRIO). The SRv6 and TSN Managers both reside in the userspace and communicate with the kernel space via the Netlink Socket interface. Figure 5 illustrates an example of *taprio-schedule* aligned with 802.1Qcw. In our example, the taprio schedule configuration for any specified SRv6 enabled TSN router is stored as a list under the tag <taprio-schedules>. Each value in the list specifies the per-port wise 802.1Qbv schedule configuration with the parameter <device> indicating the port name, whereas <sched-entries> indicates the corresponding gate control list. The gate control list consists of a list of schedule entries with <gatemask> specifying the traffic classes that are permitted for transmission within the specified time <interval>.
- **OSPF manager:** OSPFv3 sets up the stub areas of all the routers in the network and enables the exchange of link state advertisements (LSA). OSPFv3 on the user space is also used to retrieve LSAs from the OSPF database by a user space application. In this work, we enable the distribution of SRv6 SIDs between the nodes by distributing the prefix SIDs between all the nodes belonging to the same autonomous system (AS), using OSPFv3. The routing network device implementation relies on Quagga stack to enable OSPF functionality. OSPF parametrization is made using simple scripting.
- **TAPRIO-based 802.1Qbv queueing and forwarding:** The kernel space of the operating system houses the forwarding and the queuing logic responsible for basic processing of incoming packets. Any packet arriving at the incoming port of the router enters the Forwarding Logic, which is responsible for forwarding the packet to the corresponding output port. The packet is then forwarded to the queuing layer at the output port. We make use of time aware priority shaper (TAPRIO), as the queuing layer to emulate the TSN 802.1 Qbv behavior. The classifier

```
{
  "edit-config": {
    "target": {
      "running": ""
    },
    "default-operation": "none",
    "test-option": "test-then-set",
    "error-option": "rollback-on-error",
    "config": {
      "taprio-schedules": {
        "schedules": {
          "device": "dev_name",
          "sched-entries": {
            "sched-entry": [
              {
                "command": "S",
                "gatemask": "schedule[' mask']",
                "interval": "schedule[' interval']"
              },
              {
                "command": "S",
                "gatemask": "schedule[' mask']",
                "interval": "schedule['interval']"
              }
            ]
          }
        },
        "_xmlns": "urn:ietf:params:xml:ns:yang:taprio-schedules",
        "_operation": "create"
      },
      "_xmlns": "urn:ietf:params:xml:ns:netconf:base:1.0"
    }
  }
}
```

FIGURE 5 Example of TAPRIO schedule configuration in JSON

TABLE 1 DSCP traffic class mapping

DSCP name	DS field value	Traffic class mapping
CS0	0	0: Best Effort
LE	1	n/a
CS1, AF11-13	8, 10, 12, 14	1: Priority
CS2, AF21-23	16, 18, 20, 22	2: Immediate
CS3, AF31-33	24, 26, 28, 30	3: Flash - mainly used for voice signaling
CS4, AF41-43	32, 34, 36, 38	4: Flash Override
CS5, EF	40, 46	5: Critical - mainly used for voice RTP
CS6	48	6: Internetwork Control
CS7	56	7: Network Control

in the queuing layer of each output port is responsible for classification of packets to different traffic classes depending on the priority fields of the packet header. The packets are then queued into the appropriate traffic classes, where each traffic class represents a first in first out (FIFO) queue. Each traffic class is associated with a gate. Packets in a traffic class are scheduled for transmission and forwarded to the driver, only if the corresponding gate is open. The status of a gate (i.e., whether it is closed or open) at any given time instance is decided by the gate control list (GCL), which defines the time interval for which any given traffic classes are allowed for transmission.

4.2.3 | Mapping of traffic classes

According to IETF RFC 9023, mapping of a DetNet IP flow to a TSN stream is provided via the combination of a passive and an active stream identification function, used to capture the 6-tuple of a DetNet IP flow; the active stream identification function is employed to modify the Ethernet header according to the ID of the mapped TSN Stream.

QoS mapping determines how the different flow types are mapped onto the various traffic classes or hardware queues on the output of a NIC interface. SRv6 packets are handled by the IPv6 layer, which uses the differentiated services architecture (DSA) for implementing QoS provisioning of flows. DSA comprises an approach where the relative priority and Type of Services marking are derived from the QoS sensitive fields in the packet header for dynamic reservation of resources. Thus, DSA follows a per-hop behavior in each router for providing QoS guarantees, instead of an end-to-end reservation strategy. In the case of layer 2, the 3-bit VLAN priority field of the VLAN-tagged packets is utilized for determining the QoS requirements of the flow, thus, allowing a maximum of 8 traffic class combinations. With respect to layer 3, the 8-bit traffic class field in the IPv6 packet header is used to indicate the QoS requirements of the flow. The first 6 bits of the traffic class field, referred as DSCP, are used for determining the QoS mapping by the packets, thereby, allowing a maximum of 64 different traffic classes. The Diffserv RFCs recommend the following most commonly used Per-Hop Behaviors (PHB) for various DSCP combinations, as shown in Table 1.

TAPRIO exploits the priority field of the socket buffer (*sk_buff, SKB*) used by the networking stack of the Linux kernel ($skb \rightarrow priority$) in order to classify the packets to a particular traffic class. Thus, the $skb \rightarrow priority$ field needs to be modified based on the QoS fields in the packet header for mapping traffic classes to the packet priority information. Because the SKB is a kernel structure, it cannot be directly set from userspace. In order to accomplish this, we follow the *iptables*[44] approach to set the priority field of SKB, before the Qdisc has been reached.

The mangle table in the IP layer is used to modify packets. The classifier rule is appended to the POSTROUTING chain, which is invoked after the forwarding decision, just before the packet reaches the Qdisc so that the TAPRIO Qdisc can see the priority field set by the iptables rule (classifier). Table 2 shows the classifier rule for mapping the packets with DSCP value XX onto a skb priority value of Y.

TABLE 2 IPv6 rule table for traffic mapping

Chain POSTROUTING (policy ACCEPT)					
Target	**Prot**	**Source**	**Destination**	**Match**	**Action**
CLASSIFY	All	Anywhere	anywhere	DSCP match 0xXX	CLASSIFY set 0:Y

5 | EVALUATION

5.1 | Evaluation environment

In this section, we evaluate our technical approach in a modified Mininet emulator version 2.2. To this end, we extend the Linux SRv6 data plane implementation from Ventre et al[8] to support TSN 802.1Qbv based forwarding by integrating TAPRIO into this Mininet environment. The SRv6 data plane in Mininet makes use of Linux SRv6 implementation for packet forwarding. The Linux SRv6 implementation allows configuration of the segment routing Policies using the *iproute* utility. All SRv6-TSN enabled routers and hosts are emulated using linux network namespaces. Each router consists of a loopback interface and a management interface. The IPv6 address of the loopback interface forms the Node SID, while its prefix forms the prefix SID of the node. All local SIDs take the same prefix as the prefix SID. The prefix SIDs are distributed across all the nodes in the network with the help of an OSPF implementation provided by the Quagga Routing Suite.[45] This ensures that all local SIDs are reachable across the entire network. All the SRv6-TSN enabled routers communicate with the centralized controller via a management switch. The controller communicates with the SRv6-TSN enabled routers via the python based NETCONF client-server implementation. We define three types of YANG data models, namely *srv6-explicit-path*, *srv6-local-sid* and *taprio-schedules* for the configuration of SR policy, mapping of segment IDs to function and configuration of GCL for TAPRIO, respectively. We have integrated TAPRIO into the Mininet environment, as explained in our previous work.[7,32] However, in this work, we map the traffic classes of TAPRIO to the Differentiated Services Field of the IPv6 header, instead of VLAN priorities.

TAPRIO is a Linux queuing discipline that implements the IEEE 802.1Qbv Time-Aware Shaper. TAPRIO allows the configuration of a sequence of gate states, where each gate state allows outgoing traffic for a subset of traffic classes based on the notion of time slice. TAPRIO queuing discipline is supported only from Linux kernel version 5.X onwards. For this evaluation setup, we use Linux kernel version 5.4.0 installed with the latest available version of iproute2[46] package.

The implementation of the software TSN within a virtualized Mininet environment (i.e., Mininet) requires the creation of multi-queued NIC interfaces, as TAPRIO is supported only on a multi-queued NIC interface. In order to configure TAPRIO inside a virtual machine, we need to create multi-queued NIC interfaces inside Mininet, instead of the default single queue interfaces. Multiqueued NIC interfaces can be created by using virtual ethernet devices. They can act as tunnels between the network namespaces to create a bridge to a network interface in another namespace, and also can be used as standalone network devices. These devices are always created in interconnected pairs.

All experiments are conducted on an Intel NUC PC with Intel (R) Core (TM) i7-8559U CPU @ 2.70GHz, and 16 GB RAM installed with Ubuntu 20.04.1 LTS based on Linux Kernel version 5.4.0-58-generic.

5.2 | Evaluation results

The evaluation section is structured as follows. We first evaluate some of the key performance indicators (KPI) of our SRv6 over TSN-based implementation, such as SRv6 provisioning times, average delay and performance overhead. Subsequently, we proceed onto the evaluation of the overall convergence time. Note that in all emulations, all network services, the entire Mininet network, and the SDN controller/CNC (scheduler) are operating on a single PC, leading to degraded data plane performance especially when increasing network scale and Talker/Listener pairs. The data plane performance is dependent on the packet processing and monitoring operations across all the switches, similar to several other aspects such as the thread scheduling policy in effect. The results presented are indicative and our future plans include the implementation of the solution using TSN hardware prototypes.

5.2.1 | TAPRIO overhead

The objective of this experiment is to evaluate the overhead induced by the TAPRIO queuing layer under no congestion and no interference. The experiment is performed on a linear network topology with varying network dimensions connected to a pair of talker–listener. Figure 6 shows the plot of the average end-to-end latency experienced by traffic transmitted every 250μs interval of packet size 512 bytes for the cases with and without TAPRIO. According to this plot, the average end-to-end latency increases with the hop count. In addition, for any given hop count, the latency experienced with TAPRIO is higher. This is due to an additional queuing layer, which has been added to implement the time aware shaping functionality. However, the variation in latency among the various hop counts is not high, that is, 1.83 µs/hop count (without TAPRIO) and 2.03 µs/hop count (with TAPRIO). The average additional overhead experienced by the TAPRIO queuing is around 13%, as measured in our setup.

5.2.2 | Control plane-data plane interaction

In this experiment, we evaluate the time required by the SRv6 controller to convey the configuration information, such as segment routing policies, Segment ID function mapping and GCL schedules for TAPRIO configuration.

Figure 7 presents the average provisioning time required to send 5 configuration commands to any SRv6 enabled router via NETCONF protocol. The figure shows two variants of the provisioning times. The *Push Config* corresponds

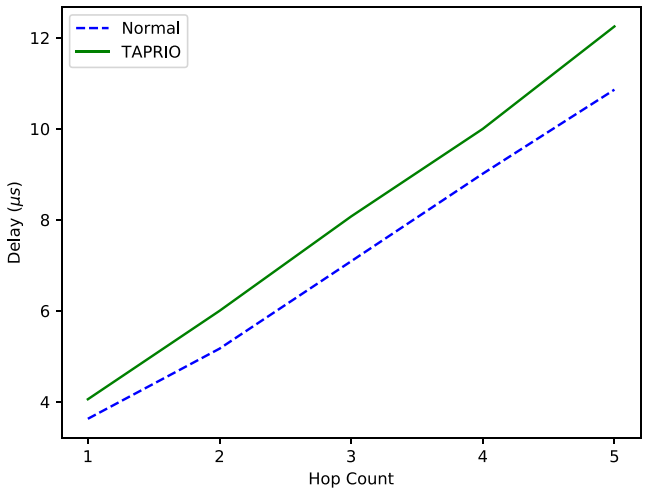

FIGURE 6 Overhead introduced by TAPRIO operation

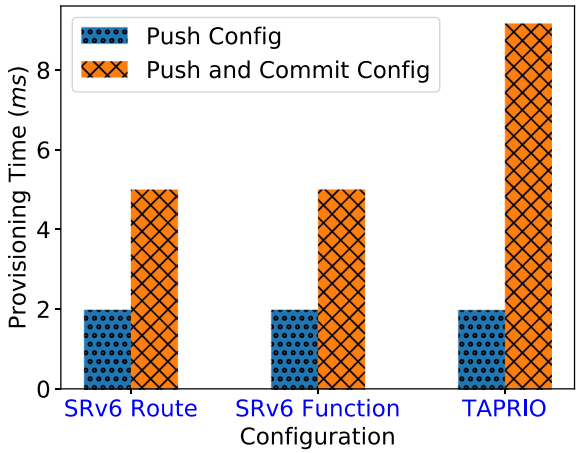

FIGURE 7 SRv6 TSN provisioning times

to the time needed for configuration information to be received and parsed by the target device; however, it does not take into consideration the time to enforce (execute) the requested configuration on the target node. The latter is taken into account by the *Push and Commit Config*. As expected, the provisioning time for all three types of configurations namely SRv6 Route, SRv6 Function, and TAPRIO for *Push Config* is the same (i.e., around 2 ms). With respect to *Push and Commit Config*, both SRv6 Route and the SRv6 Function experience a provisioning time of around 5ms. Instead, this delay is increased in the case of TAPRIO. The additional overhead for TAPRIO is due to the shell invocation for enforcing each of the GCL schedule configurations, unlike the case of configuring SRv6 Route and SRv6 Function, in which configurations are enforced using the Netlink socket API without the need for a shell invocation.

5.2.3 | SRv6 encoding and decoding overhead

The objective of this experiment is to evaluate the overhead of the segment routing encoding and decoding operations. The topology under consideration is a linear network topology consisting of two TSN enabled SR routers. These routers are connected to a pair of talker and listener via an edge router which is also SR enabled, whereas the talker sends a 512-byte packet stream. We measure the time required by the edge router to insert incoming IPv6 packets with SRH consisting of a list of segments to be traversed, before reaching the destination node (termed as the SRv6 encoding operation). We further report the time needed by the intermediate SRv6 enabled routers to identify the corresponding segment from the segments left field of the header and forward the packet to the corresponding output port (termed as SRv6 decoding operation).

According to Table 3, the SRv6 encoding incurs a higher overhead of around 31.65% (percentage overhead in relation to normal IPv6 forwarding), since the encoding operation involves the insertion of new header information into the IPv6 packet. Instead, the SR decoding operation experiences an overhead of 14.6%, due to the additional overhead involved in decoding the forwarding information from the packet header.

5.2.4 | SRv6 forwarding overhead

The objective of this experiment is to evaluate SRv6 forwarding plane on our emulation environment. The topology under consideration is a linear topology of varying hop counts. Figure 8 shows the average delay for forwarding the packets of 512 bytes over normal IPv6 and SRv6 forwarding planes. We observe that the SRv6-based forwarding experiences a higher transmission delay, due to the decoding of the forwarding information from the SRH of the packet header.

TABLE 3 SRv6 operation overhead

	Normal IPv6 forwarding	SRv6 encoding	SRv6 decoding
Delay (μs)	1.7613	2.3188 (31.65 %)	2.0187 (14.06 %)

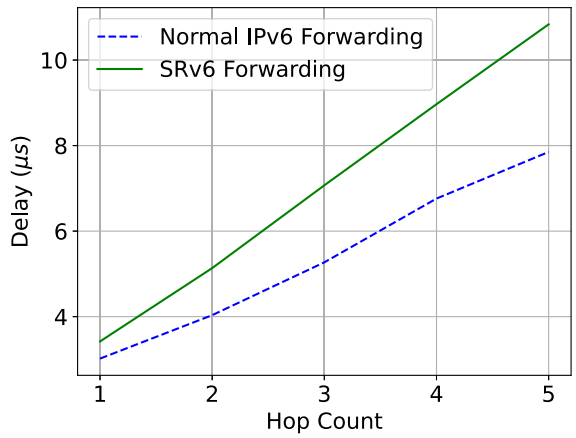

FIGURE 8 SRv6 forwarding overhead

5.2.5 | Controller-to-switch configuration overhead

In this experiment, we measure the time needed by the SDN controller for the configuration of multiple network nodes in parallel via a NETCONF interface. Figure 9A shows the controller provisioning time (with *Push Config*) and the CPU/memory utilization of the SDN controller. We quantify the configuration overhead for the insertion of 100 forwarding rules per ingress node (which perform the actual path encapsulation). Figure 9B indicates that increasing the number of nodes managed by a single controller yields increased provisioning time, as well as increased resource utilization by the centralized controller. For fully centralized SDN-based solutions, this is a well-known issue. Especially for large network topologies, requirements such as fast rerouting of time-critical flows are difficult to be satisfied. Instead, our SR-based hybrid scheme reduces the number of controller-switch interactions, since forwarding rules need to be populated only onto the access switches.

5.2.6 | Convergence time

The objective of this experiment is to assess the convergence of OSPF and SRv6 (combined with OSPF). The topology under consideration is a ring topology consisting of five SR-enabled TSN routers. Figure 10A shows the OSPF message sequence as monitored from an arbitrary port in the topology. The blue lines correspond to the time sequence during normal operation. In the normal case, each node sends an OSPF *hello message* at every 1-s interval (tunable from OSPF

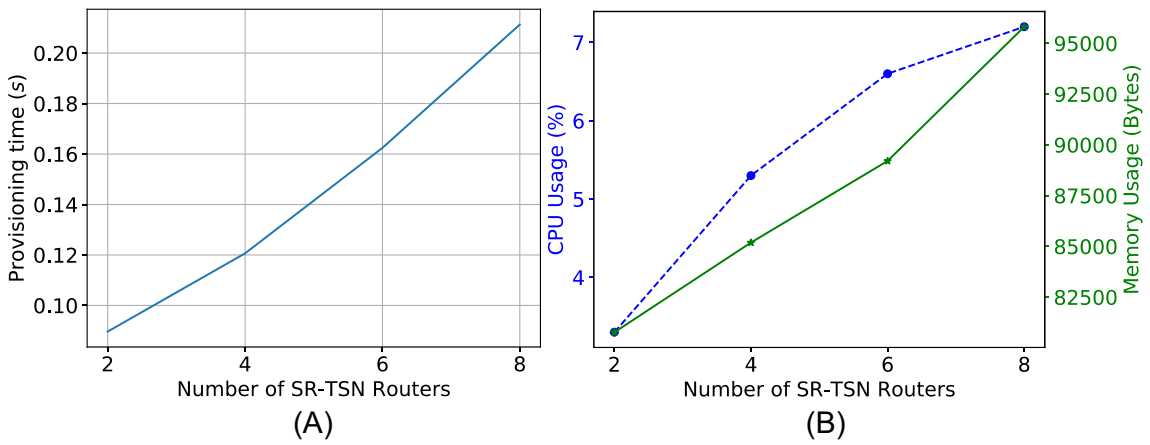

FIGURE 9 (A) Controller configuration time. (B) CPU and memory utilization of Controller

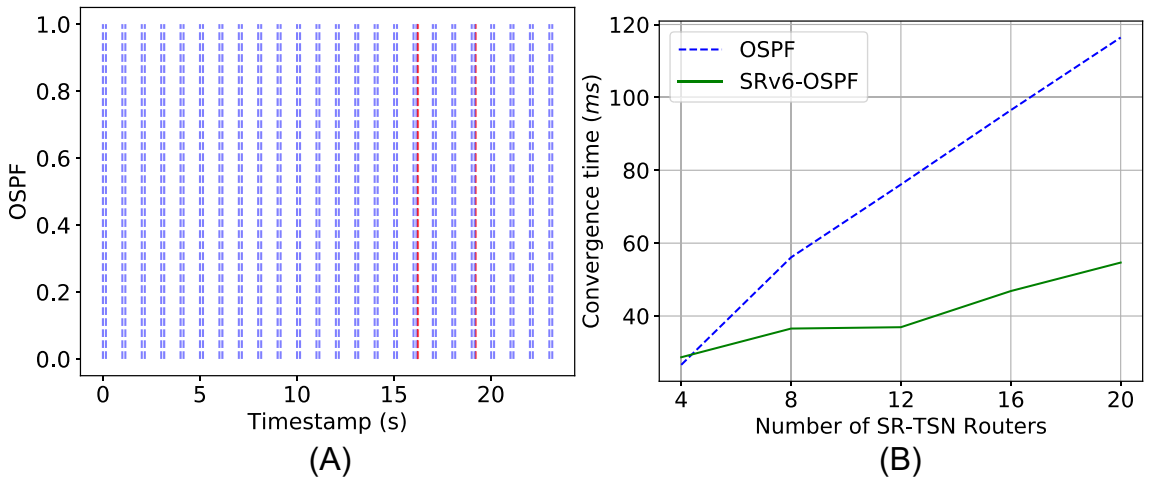

FIGURE 10 (A) OSPF messaging. (B) Segment Routing convergence time

configuration files). In our scenario, we observe two OSPF *hello messages* for every 1 s, since each router is connected to two immediate neighbors. In the event of a failure or topology change, a link-state update message is sent over an IPv6 multicast group to inform all routers in the network of the topology change. Upon the notification of this topology change, each router updates its OSPF database and sends a link-state acknowledgment message confirming the reception of the topology update event. It should be noted that OSPF experiences a much lower convergence time compared with other routing protocols, such as the spanning-tree variants. This is due to the fact that each node recovers from the failure individually by executing the shortest-path algorithms. However, the network convergence time is high especially in the case of larger network topologies, because the topology change event has to be conveyed to all nodes in the network.

This phenomenon can be observed in Figure 10B, which shows the convergence time of segment routing for a diverse range of network sizes. We measure the convergence time, which is defined as the time required to re-establish the flows in the event of a link failure. We use the ring topology with a link latency of 5ms and varying network sizes in order to obtain the plot of convergence time for OSPF and SRv6-OSPF, respectively. In the case of OSPF, packets are forwarded using the routing paths determined by the OSPF algorithm. SRv6-OSPF, on the other hand, steers packets as a list of segments, with OSPF used to distribute forwarding information to reach individual segments. According to Figure 10B, OSPF experiences a lower convergence time than SRv6 for smaller network sizes consisting of 4 SRv6-TSN devices. This stems from the fact that the communication overhead for sending the configuration information from a centralized controller via NETCONF over SSH connection is higher than the overhead for distributing the failure information over smaller topologies. With increasing network sizes, it is evident that the SRv6-OSPF case experiences a lower convergence time. This is because, in the case of SRv6 during a topology update event, the controller needs to notify only the edge routers. Furthermore, it should be noted that in our experiments, even though SRv6 forwarding uses OSPF to distribute the adjacency SIDs, the recovery time has less dependency on OSPF convergence time. This is due to the fact that the routing information is now encoded as a list of adjacency segments and packets are re-routed over those segments which are unaffected by the failure. We note, however, that in case SIDs map to nodes different performance results are expected due to OSPF operations.

5.3 | Convergence time investigation

All the experiments, throughout this section, are performed using three different types of traffic classes, namely, time-triggered traffic (Priority 7), audio video bridging traffic (Priority 5), and best-effort traffic (Priority 0). For each flow, the traffic class is randomly selected from the above set, originating from Talker-Listener pairs connected randomly to different nodes in the network.

5.3.1 | Flow priority aware routing heuristic

This experiment aims at evaluating our proposed routing heuristic (i.e., FPAR) in comparison to shortest path routing. The evaluation is performed by simulating the scenario of allocating multiple flows for an 8-ary fat-tree data center network topology. Maximum in-class interference is defined as the maximum number of traffic flows of same priority flowing through any given link in the network and is used as a KPI for evaluating the benefit of our routing heuristic.

Figure 11 shows the comparison of maximum in-class interference for all three traffic classes while increasing the number of flows for both FPAR and traditional shortest-path routing (denoted as SP).

From the plot, it is evident that the maximum in-class interference increases with the increasing number of flows for all the traffic types. However, the maximum in-class interference for any given number of flows with FPAR is significantly lower than shortest path routing. A lower value of maximum in-class interference leads to lower complexity when scheduling the time-triggered flows. That is because a lower value of maximum in-class interference leads to an increased probability of computing a non-overlapping path for newly arriving time-critical flows with strict end-to-end deadlines.

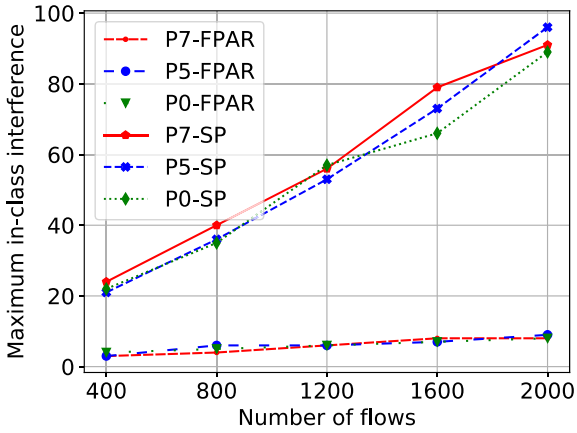

FIGURE 11 Max in-class interference for various routing schemes

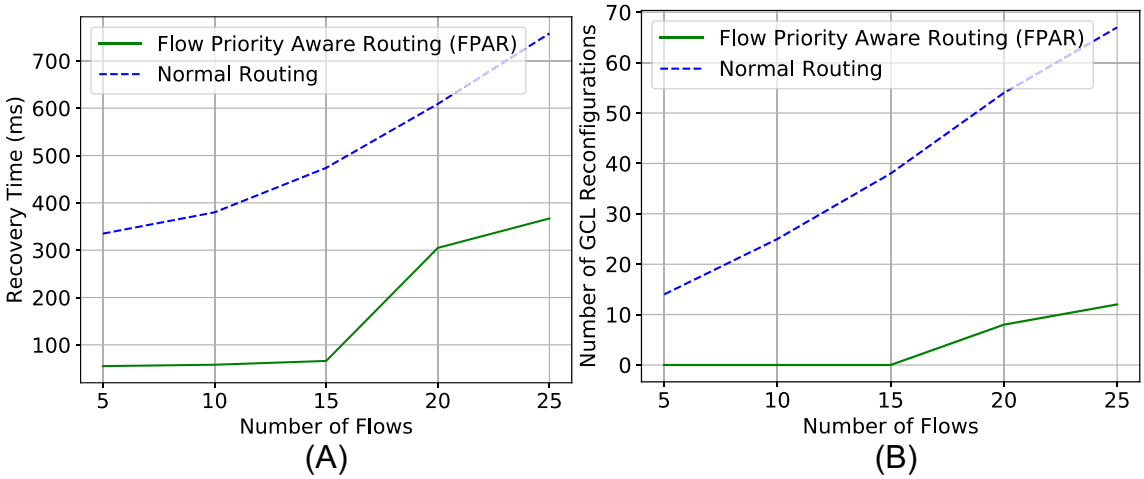

FIGURE 12 Increasing the number of flows: (A) Recovery time; (B) Number of GCL re-configurations (Random network topology with 25 nodes and 50 edges)

5.3.2 | Network recovery with TSN, FPAR, and source routing

In this experiment, we evaluate the integrated FPAR and scheduling framework. The path configuration is based on source routing, according to our previous work in Kumar et al[7] and is depicted for comparison reasons. The experiments are conducted on a random network topology with Talker and Listener pairs connected to the core TSN network in a random fashion with a packet size of 1440 bytes. Under FPAR, the scheduler is only invoked over the minimal subgraph required to identify the GCLs on those paths that are affected by the failure, instead of the complete network topology graph. The normal scenario corresponds to the case where the scheduler is applied over the entire topology and the entire set of flows.

Figure 12 shows the average network recovery time and the GCL configurations, due to a random link failure for a varying number of flows through 25 switches and 50 links on a random network topology for a randomly chosen link failure. According to Figure 12A, the average recovery time increases with the increasing number of flows, since the execution time of a scheduler depends on the number of flows to be scheduled. The proposed FPAR technique allows to minimize the number of overlapping flows, thereby, opening up the opportunities for running the scheduler only on a minimal subset of flows over a minimal connected subgraph. This leads to a significant reduction of the recovery time, due to the reduced GCL reconfigurations required, as shown in Figure 12B.

5.3.3 | Network recovery with TSN, FPAR, and SRv6

In this experiment, we exploit SRv6 for generating explicit routing paths. Figure 13A depicts TSN recovery time for a diverse number of talker–listener pairs. The topology under test is a random Erdos-Rényi network topology consisting of five nodes and 10 links. The plot shows a similar response, as that of Figure 12A with the proposed FPAR techniques showing reduced convergence time compared with normal routing. It can be observed, however, that SRv6 experiences higher recovery time compared with the source routing case. This increase in the recovery time is attributed to the increased stress on the Mininet environment, while processing L3 SRv6 decoding and forwarding operations (plus TSN queuing), unlike the case of source routing, where all the packets are processed at L2 of the protocol stack (plus TSN queuing).

Figure 13B shows the network recovery time for a diverse number of nodes. A fixed number of 3 pairs of Talkers and Listeners are chosen to generate randomly selected traffic types. In the case of typical shortest path routing, the average number of GCL configurations increases with the increase in the network size. In the case of FPAR, average GCL configurations decreases for a fixed number of flows with increasing network sizes up to a certain range, since the probability of an overlapping allocation of flows is minimized due to the presence of more alternative paths. The number of GCL re-configurations decreases, as the scheduler is invoked only on the subgraph containing the problematic/overlapping flows.

5.3.4 | Network recovery under large-Scale TSN

Recall that all the previous tests were conducted using the integrated Mininet solution. Note that beyond 25 (emulated) switches the Mininet environment is not stable in our system. To mitigate this problem and assess issues of scale, we resort to python-based simulations where we quantify the GCL reconfigurations required in the case of a failure. Figure 14A illustrates the average number of GCL configurations required to recover all the affected flows, due to a random link failure in an Erdos-Rényi random network graph consisting of 10 TSN switches for a varying number of randomly selected flows. The red line in the plot illustrates the maximum number of GCL configurations in the network, which can be deduced when all the ports used in the network need to be reconfigured. As expected, the average number of required GCL configurations increases for normal routing, when increasing the number of flows. Although an increase is also observed in the GCL configurations with FPAR (in relation to the number of flows), FPAR overall requires significantly fewer GCL configurations compared with normal routing.

Figure 14B shows the average number of GCL configurations required during a TSN failure event on a network of diverse size with a fixed number of flows. The topology under test is a random Erdos-Rényi network graph consisting of varying network dimensions. A fixed number of 100 pairs of talkers and listeners chosen randomly are used to generate randomly selected traffic types. In the case of normal routing, the average number of GCL configurations increases with the increase in the network size. However, in the case of FPAR, average GCL configurations decreases for a fixed

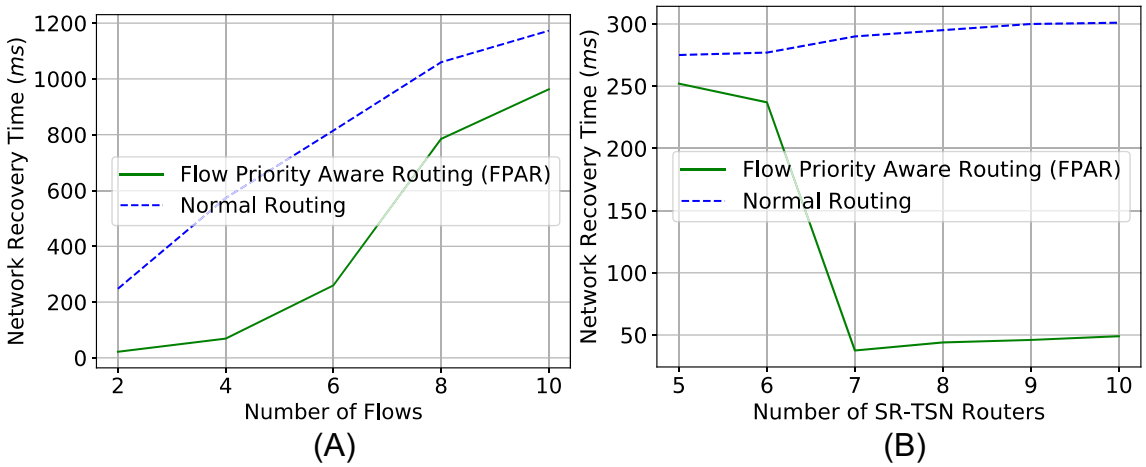

FIGURE 13 SRv6 TSN recovery time vs. (A) number of flows (5 nodes); (B) number of nodes (3 flows)

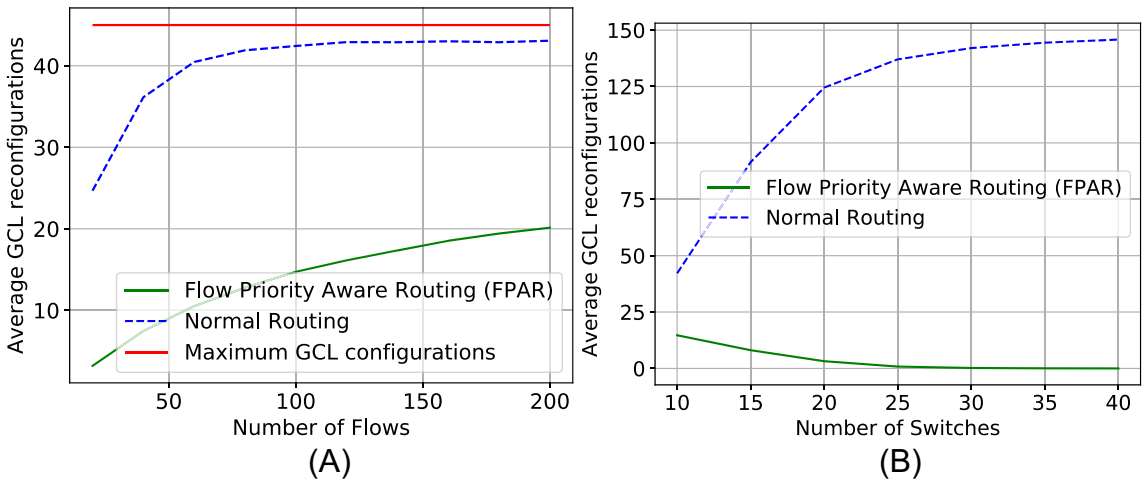

FIGURE 14 GCL reconfigurations vs. (A) number of flows (10 TSN nodes); (B) number of switches (100 flows)

number of flows with increasing topology sizes as the scheduler is invoked only on the subgraph containing the problematic/overlapping flows.

5.4 | Discussion points

In our approach, we consider that all the network devices are IEEE TSN 802.1Qbv and SRv6 capable. More specifically, the technical approach, which we rely upon, performs integration of SRv6 with 802.1Qbv TSN in the control plane, where inside the SDN controller the control application is deciding about routing and scheduling. Although this approach has some direct benefits, such as ease of implementation due to the fact that existing Linux tools can be used, there are a number of shortcomings identified.

For example, we are not considering relevant synchronization issues for large-scale setups, which in practice may inhibit 802.1Qbv adoption for such scenarios. In our future work, we plan to investigate multi-domain TSN environments to tackle such issues for large-scale networks. In the same context, we have also not investigated aspects related to DetNet Flow Aggregation, as considered by IETF RFC 8939. Furthermore, although we are aligned with IETF RFC 9023 and we also consider a mapping function between IP and Ethernet, our mapping function is only used to map the relevant QoS characteristics, and, as such, we are not following 802.1CB related identifiers.

By means of comparison, other techniques such as P802.1Qch (Cyclic Queuing and Forwarding) and 802.1Qcr Asynchronous Traffic Shaper (ATS) besides 802.1Qbv are left for future work. Regarding the operation of the FPAR policy our future plans include performance and overhead analysis and comparison with other Fast Reroute solutions (e.g., loop-free alternate [LFA]; IETF RFC5286) and variations such as topology-independent loop-free alternate (TI-LFA), which are applicable to segment routing for constructing backup paths.

6 | CONCLUSIONS

In this work, we elaborated on an SDN-based approach for low-overhead network updates over TSN, when coupled with IPv6 segment routing. In this respect, we exploited the concept of TSN subgraphs in order to quickly reschedule the flows traversing the updated/problematic area and proposed a TSN-aware routing heuristic to minimize convergence time. The methodology and mechanisms considered are applicable when dynamic network updates and/or failure event handling are required. We described the TAPRIO (802.1Qbv) and SRv6 integration, as well as our implementation experience in Linux and how the environment can be integrated into Mininet.

Besides addressing the issues explained in Section 5.4, our future work will further encompass tighter integration between TSN and SRv6, following the latest IETF and IEEE principles of operation; enhancements of the proposed recovery algorithm in order to cope with multiple link/node failures occurring at different parts of large network

topologies; and investigation of better strategies for transforming the routing path to the segment list, such that explicit path routing can be defined with fewer segments. In addition, the proposed control plane models can be extended for the needs of multi-domain TSNs with the border gateway switches of each domain being responsible for encoding the path information within the respective domain. Finally, our approach can be further enhanced with other algorithmic variants such as fault resilient routing and scheduling strategies for better handling the case of dynamically changing operational requirements.

ACKNOWLEDGEMENTS

Open Access funding enabled and organized by Projekt DEAL.

DATA AVAILABILITY STATEMENT

Research data are not shared.

ORCID

Gagan Nandha Kumar https://orcid.org/0000-0002-1174-5847
Kostas Katsalis https://orcid.org/0000-0001-9282-6747

REFERENCES

1. Gavriluţ V, Zhao L, Raagaard ML, Pop P. AVB-aware routing and scheduling of time-triggered traffic for TSN. *IEEE Access*. 2018;6:75229-75243. doi:10.1109/ACCESS.2018.2883644
2. Schweissguth E, Danielis P, Timmermann D, Parzyjegla H, Mühl G. ILP-based joint routing and scheduling for time-triggered networks. In: RTNS; 2017:8-17.
3. Pahlevan M, Tabassam N, Obermaisser R. Heuristic list scheduler for time triggered traffic in time sensitive networks. *ACM Sigbed*. 2019;16(1):15-20.
4. Jyothi SA, Dong M, Godfrey PB. Towards a flexible data center fabric with source routing. In: ACM SIGCOMM; 2015:1-8.
5. Filsfils C, Previdi S, Ginsberg L, Decraene B, Litkowski S, Shakir R. Segment Routing Architecture. RFC 8402; 2018.
6. Ventre PL, Salsano S, Polverini M, et al. Segment routing: a comprehensive survey of research activities, standardization efforts, and implementation results. *IEEE Commun Surv Tutorials*. 2021;23(1):182-221.
7. Kumar GN, Katsalis K, Papadimitriou P, Pop P, Carle G. Failure handling for time-sensitive networks using SDN and source routing. In: 2021 IEEE 7th International Conference on Network Softwarization (NetSoft); 2021:226-234.
8. Ventre PL, Tajiki MM, Salsano S, Filsfils C. SDN architecture and southbound APIs for IPv6 segment routing enabled wide area networks. *IEEE Trans Netw Serv Manag*. 2018;15(4):1378-1392.
9. Sambo N, Fichera S, Sgambelluri A, Fioccola G, Castoldi P, Katsalis K. Enabling delegation of control plane functionalities for time sensitive networks. *IEEE Access*. 2021;9:136151-136163.
10. Shreedhar M, Varghese G. Efficient fair queuing using deficit round-robin. *IEEE/ACM Trans Networking*. 1996;4(3):375-385.
11. Nasrallah A, Thyagaturu AS, Alharbi Z, et al. Ultra-low latency (ULL) networks: the IEEE TSN and IETF DetNet standards and related 5G ULL research. *IEEE Commun Surv Tutorials*. 2019;21(1):88-145.
12. Steiner W, Craciunas SS, Serna Oliver R. Traffic planning for time-sensitive communication. *IEEE Commun Stand Mag*. 2018;2(2):42-47. doi:10.1109/MCOMSTD.2018.1700055
13. Craciunas SS, Serna Oliver R, Chmelík M, Steiner W. Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks. In: RTNS; 2016:183-192.
14. Zhao L, Pop P, Craciunas SS. Worst-case latency analysis for IEEE 802.1 Qbv time sensitive networks using network calculus. *IEEE Access*. 2018;6:41803-41815. doi:10.1109/ACCESS.2018.2858767
15. Maxim D, Song Y-Q. Delay analysis of AVB traffic in time-sensitive networks (TSN). In: RTNS; 2017:18-27.
16. Reusch N, Zhao L, Craciunas SS, Pop P. Window-based schedule synthesis for industrial IEEE 802.1Qbv TSN networks. In: 2020 16th IEEE International Conference on Factory Communication Systems (WFCS); 2020:1-4. doi:10.1109/WFCS47810.2020.9114414
17. Laursen SM, Pop P, Steiner W. Routing optimization of AVB streams in TSN networks. *ACM Sigbed*. 2016;13(4):43-48.
18. Nayak NG, Duerr F, Rothermel K. Routing algorithms for IEEE 802.1 Qbv networks. *ACM SIGBED*. 2018;15(3):13-18.
19. Nayak NG, Dürr F, Rothermel K. Time-sensitive software-defined network (TSSDN) for real-time applications. In: RTNS; 2016.
20. Pop P, Raagaard ML, Gutierrez M, Steiner W. Enabling fog computing for industrial automation through Time-Sensitive Networking (TSN). *IEEE Commun Stand Mag*. 2018;2:55-61.
21. Atallah AA, Hamad GB, Mohamed OA. Fault-resilient topology planning and traffic configuration for IEEE 802.1 Qbv TSN networks. In: IOLTS IEEE; 2018:151-156.
22. Mas-Machuca C, Musumeci F, Vizarreta P, et al. Reliable control and data planes for softwarized networks. *Guide to disaster-resilient communication networks*: Springer; 2020:243-270.
23. Bianchi G, Pontarelli S, Bonola M, Cascone C, Sanvito D, Capone A. Method of handling data packets through a conditional state transition table and apparatus using the same. US Patent 10,708,179; 2020.

24. von Tüllenburg F, Pfeiffenberger T. Layer-2 Failure Recovery Methods in Critical Communication Network. ICNS; 2016.
25. Saldamli G, Mishra H, Ravi N, Kodati RR, Kuntamukkala SA, Tawalbeh L. Improving link failure recovery and congestion control in SDNs. 2019 10th International Conference on Information and Communication Systems (ICICS). 2019:30-35.
26. Sgambelluri A, Giorgetti A, Cugini F, Paolucci F, Castoldi P. Openflow-based segment protection in ethernet networks. *IEEE/OSA J Opt Commun Netw*. 2013;5(5):1066-1075.
27. Papagianni C, Papadimitriou P, Baras JS. Towards reduced-state service chaining with source routing. 2018 14th International Conference on Network and Service Management (CNSM). 2018:438-443.
28. Papadopoulos K, Papadimitriou P. Leveraging on source routing for scalability and robustness in datacenters. IEEE 5GWF; 2019.
29. Bozakov Z, Papadimitriou P. Towards a scalable software-defined network virtualization platform. In: IEEE NOMS; 2014:1-8.
30. Hari A, Lakshman TV, Wilfong G. Path switching: reduced-state flow handling in SDN using path information. In: ACM CONEXT; 2015.
31. Guo C, Lu G, Wang HJ, et al. SecondNet: a data center network virtualization architecture with bandwidth guarantees. In: ACM CONEXT; 2010.
32. Kumar GN, Katsalis K, Papadimitriou P. Coupling source routing with time-sensitive networking. In: 2020 IFIP Networking Conference (Networking); 2020:797-802.
33. Bashandy A, Filsfils C, Previdi S, Decraene B, Litkowski S, Shakir R. Segment Routing with the MPLS Data Plane. RFC 8660; 2019.
34. Filsfils C, Dukes D, Previdi S, Leddy J, Matsushima S, Voyer D. IPv6 Segment Routing Header (SRH). RFC 8754; 2020.
35. Filsfils C, Camarillo P, Leddy J, Voyer D, Matsushima S, Li Z. Segment Routing over IPv6 (SRv6) Network Programming. RFC 8986; 2021.
36. Bhatia R, Hao F, Kodialam M, Lakshman TV. Optimized network traffic engineering using segment routing. In: 2015 IEEE Conference on Computer Communications (INFOCOM), 2015:657-665. doi:10.1109/INFOCOM.2015.7218434
37. Aubry F, Lebrun D, Vissicchio S, Khong MT, Deville Y, Bonaventure O. SCMon: leveraging segment routing to improve network monitoring. In: INFOCOM; 2016.
38. Aubry F, Vissicchio S, Bonaventure O, Deville Y. Robustly disjoint paths with segment routing. In: Proceedings of the 14th International Conference on Emerging Networking Experiments and Technologies; 2018:204-216. doi:10.1145/3281411.3281424
39. Chen M, Geng X, Li Z. Segment Routing (SR) Based Bounded Latency, Internet-Draft draft-chen-detnet-sr-based-boundedlatency-01, Internet Engineering Task Force; 2019. Work in Progress.
40. Wang X, Dai J, Liu J, Zhang F. DetNet Data Plane: IEEE 802.1 Time Sensitive Networking over SRv6, Internet-Draft draft-wang-detnet-tsn-over-srv6-04, Internet Engineering Task Force; 2021. Work in Progress.
41. Thubert P, Yang F. IPv6 Options for DetNet, Internet-Draft draft-pthubert-detnet-IPv6-hbh-07, Internet Engineering Task Force; 2022. Work in Progress.
42. NetworkX x2014; NetworkX documentation—networkx.org. [Accessed 16-Apr-2022]. https://networkx.org/
43. Lebrun D, Bonaventure O. Implementing IPv6 segment routing in the Linux kernel Proceedings of the Applied Networking Research Workshop; 2017:35-41.
44. Purdy. *Linux iptables Pocket Reference: Firewalls, NAT & Accounting*. O'Reilly Media, Inc.; 2004.
45. Jakma P, Lamparter D. Introduction to the quagga routing suite. *IEEE Netw*. 2014;28(2):42-48.
46. Kuznetsov A, Hemminger S. Iproute2: a collection of utilities for controlling TCP/IP networking and traffic control in Linux.

How to cite this article: Nandha Kumar G, Katsalis K, Papadimitriou P, Pop P, Carle G. SRv6-based Time-Sensitive Networks (TSN) with low-overhead rerouting. *Int J Network Mgmt*. 2023;33(4):e2215. doi:10.1002/nem.2215