Coupling Source Routing with Time-Sensitive Networking

Gagan Nandha Kumar*, Kostas Katsalis*, Panagiotis Papadimitriou†
*Huawei Technologies Duesseldorf Gmbh, Munich, Germany
†Department of Applied Informatics, University of Macedonia, Greece

***Abstract*—We discuss source routing over time-sensitive networks (TSN) as means to create explicit forwarding paths. The proposed source routing scheme over TSN can be used in multiple scenarios and complex topologies, with the aim to satisfy strict latency requirements of modern network applications. In this respect, we present the implementation of this technique in Mininet, exploiting the TAPRIO interface, together with SDN control and routing header encoding. We discuss evaluation results for scenarios, where source-routing and TSN scheduling are concurrently applied to protect high-priority traffic.**

***Index Terms*—IEEE TSN, deterministic networking, source routing, QoS.**

I. Introduction

The wide deployment of communication networks in many diverse operational environments, spanning from mobile computing and datacenter environments to automotive and industrial networks, has radically increased divergence by means of operational requirements. Different applications require diverse performance and QoS guarantees, not only in terms of throughput, but also in terms of predictable latency and jitter, such as in the case of industrial networks.

IEEE 802.1 TSN family of standards aims at providing an application-agnostic solid underpinning for performance guarantees for all Ethernet-based communications. A number of IEEE 802.1Q amendments, such as 802.1Qbv (scheduled traffic) and 802.1Qbu (preemption), are enhancing traditional 802.1 forwarding plane operations, while others are enhancing control plane aspects, such as 802.1Qcc and 802.1Qdd for the centralized configuration model and for the fully distributed, respectively. Although several research studies have proposed and analyzed solutions for the TSN scheduling problem [1], only recently TSN scheduling and routing have been considered under a common optimization framework [2], [3].

In this work, we focus on the joint TSN scheduling and routing problem. In this respect, we leverage on the approaches presented in [2], [3] and analyze the effects of using a source routing scheme over TSN-aware network fabric in order to define explicit forwarding paths. When the forwarding decision is fixed (*e.g.,* when operating over spanning trees or variations), in the case of complex Talker-Listener relationships and complicated traffic patterns, may not always be the case that an optimal scheduling scheme exists to admit the entire flow set. As a joint routing-scheduling scheme is able to broaden the solution space, we aim at designing, implementing and evaluating a conjoint routing plus scheduling TSN system, based on the source routing paradigm, able to provide latency guarantees, while also improve network efficiency.

Annex to ISBN 978-3-903176-28-7© 2020 IFIP

A source routing scheme for TSN is still compliant with a control plane, such as 802.1Qcc. However, for the forwarding decision no STP, MSTP, RSTP protocol is required, since the forwarding decision is encoded in each packet, leading to a nearly stateless TSN dataplane. In more detail, in our approach (i) we exploit an advanced centralized SDN controller for encoding path information on the edge switches; (ii) one-time configuration of the path decoding process is required at each TSN-aware switch; (iii) the path is encoded into Ethernet frames, and, as such, no IP encapsulation is required. We evaluate this TSN-aware source routing based fabric in small-scale network topologies with varying workload characteristics and strict delay requirements. Our implementation is based on Mininet, OpenvSwitch (OVS), and Time Aware Priority Shaper (TAPRIO) open-source solutions for realizing source routing over TSN. TAPRIO is a Linux-based queuing discipline implementation of the scheduling state machine defined by IEEE 802.1Qbv, which allows configuration of a sequence of gate states, where each gate state allows outgoing traffic for a subset of traffic classes. Our contributions are as follows:

- We employ source routing for a TSN-aware nearly stateless data plane.
- We provide an implementation of this source routing fabric in a TSN-aware TAPRIO-based virtual environment.
- We study the performance of TAPRIO on physical and virtual interfaces.
- We describe a header format that can be used to realize source routing in pure Ethernet frames.
- We present evaluation results when applying concurrently source routing and TSN reconfiguration in order to satisfy strict latency requirements.

In Section II, we provide background information on TSN and source routing. In Section III, we discuss our technical approach, as well as implementation details of source routing over virtual TSN. In Section IV, we present evaluation results. In Section V, we summarize our findings and outline directions for future work.

II. Background Information

A. Background on TSN

IEEE 802.1 TSN family of standards is used to provide deterministic performance by means of delay and jitter for

challenging applications, such as in the case of industrial and in-vehicle networks. For example, 802.1Qbv is incorporating the operation of transmission gates controlling precisely the time packets from a queue that are allowed to be served; 802.1Qbu is introducing a preemption step in the forwarding plane; 802.1Qci is used to apply per-stream filtering and policing. A comprehensive study of the TSN can be found in [1], whereas SMT based scheduling for TSN is presented in [4]. Note that the actual end-to-end performance depends not only on the scheduling principle in effect, but also on the path selection decision. Authors in [2] explore how the routing of time-triggered flows affects their schedulability. Authors in [3] present various Integer Linear Program (ILP) formulations that solve the combined problem of routing and scheduling time-triggered traffic, while following the SDN-based paradigm.

From the standards perspective, IEEE 802.1aq specifies the Shortest Path Bridging (SPB) principle. In 802.1Qcc amendment, many path control protocols are supported, such as STP, MSTP, RSTP (spanning tree variations), and SPB. 802.1Qca can be used for path control and reservation, while it relies on IS-IS to carry control information. IEEE 802.1Qca can be used to define Explicit Trees, an interesting feature towards implementing joint scheduling and routing strategies. This is achieved using a PCE element to find the optimal path; however, the propagation of the control information is passed with IS-IS TLVs in a distributed manner. For real deployments, the operation of IS-IS introduces high overhead, especially in the case of network failures or re-configuration, as the network state has to be maintained at each hop. In contrast, source routing operates over a nearly stateless data plane, as the path information is encoded at each packet a priori at the source.

B. Source Based routing

Source routing comprises a viable approach to the reduction of forwarding state [5], [6], [7]. This state reduction can yield significant switch TCAM savings, allowing for cheaper switching hardware. In principle, source routing encodes the path into each packet header, enabling switches to forward packets using a minimal number of (nearly static) flow-independent forwarding entries. In particular, the path is encoded as a set of labels, which correspond (or may even match) to the sequence of switch ports that each packet needs to traverse. As such, there is no need for the switches to maintain L2 or L3 forwarding entries to all destinations within the network, saving a significant amount of Ternary Content Addressable Memory (TCAM) space.

One requirement, though, is to point each switch to the encoded label that corresponds to the next output port in the path. A straightforward way to implement such a pointer is to allocate header space (*e.g.,* a dedicated header field). This pointer will need to be incremented, as the packet is forwarded along the path. This requires an additional action on the packet by each switch, before sending out each packet to the output port. Alternatively, the pointer can be implemented using the TTL field, obviating the need to perform an additional

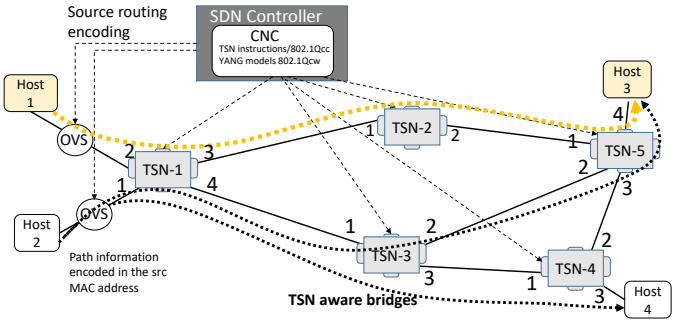

Fig. 1: Example of source routing over TSN.

pointer update; the next label can be identified by the deviation between the initial and the current TTL value [7], [5].

The required set of labels can be encoded to header fields, such as source/destination MAC address and IP(v6) address. In the case of paths with a large number of hops, additional header space can be utilized, such as VLAN and MPLS headers, at the expense of extra transmission overhead (which is more critical for smaller packets). Furthermore, source routing requires the insertion of the set of labels into the packet header, which will be handled by the switch fabric. The process of source routing can be facilitated by the deployment of a network controller (*e.g.,* OpenFlow), which will be responsible for the path computation and the population of path insertion entries into the edge switches. An implementation of a source routing scheme using OpenFlow is described in [7].

III. Technical Approach

In our approach, we exploit a SDN control plane supporting the relevant 802.1Qcc operations regarding, for example, scheduling configuration actions, while also used to realize the source routing paradigm. Using source routing over a TSN enabled forwarding plane, we are able to design optimal policies that are jointly considering the TSN scheduling and routing problem by defining explicit paths. In contrast to 802.1Qca, source routing obviates the need to keep the network state at each network bridge, so both the configuration and operational overhead is negligible. Since in source routing the path information is encoded into each packet, the bridge at each hop merely needs to perform the decoding action for each packet in order to identify the next hop. Furthermore, there is no need to perform tedious and awkward TCAM updates based on a centralized (*e.g.,* OpenFlow or NETCONF) or distributed protocol (*e.g.,* IS-IS). The restriction, in this case, is that the path computation module (which determines the path each packet will follow) needs to be aware of the entire topology; nevertheless, this difficulty can be circumvented by a SDN controller, which can easily retrieve this information (*e.g.,* through LLDP).

A. Source Routing Path Encoding

In source routing, the path selection decision for each packet is decided a priori before transmission, while the path information is encapsulated within each packet. New techniques, such

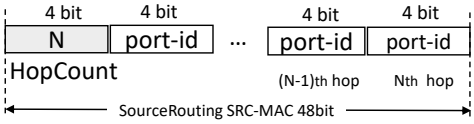

Fig. 2: Source routing MAC header encoding format.

as segment routing, exploit the source routing paradigm and encapsulate the path-segment information either over a MPLS or IPv6 dataplane. In our approach, since we are interested in pure layer 2 TSN operations, similarly to [7], we encode the path information inside the MAC header. In particular, the source MAC address is manipulated in order to encode the sequence of bridge output ports, along the computed path. In order to elaborate the concept, we use an exemplary system architecture for a network of five TSN switches and four hosts, as shown in Fig. 1. Host 1 (Talker) sends traffic to Host 3 (Listener), whereas Host 2 (Talker) sends traffic to both Hosts 3 and 4 (Listeners). Instead of using SPB, STP, or variations such as MSTP, RSTP, we seek to steer the traffic as follows; f_{13} follows the path $1 \rightarrow 2 \rightarrow 5$, f_{23} traverses the path $1 \rightarrow 3 \rightarrow 4 \rightarrow 5$, while f_{24} follows the path $1 \rightarrow 3 \rightarrow 4$. At each switch TSN operations, such as 802.1Qbv gates control, are enabled. At each hop in order to steer the packet in a specific direction, we specify the forwarding action based on which egress port to use. For example, if after TSN-1 the next hop is TSN-2, then port 3 needs to be used as the egress at TSN-1. A path encoding module is used to encode the entire path within each packet. We specifically encode the path within the source MAC address; Fig. 2 illustrates the encoding format. Each egress port is encoded using 4 bits (N^{th} hop bits 0 to 3, $(N-1)^{th}$ hop bits 4 to 7, padding for the unused, etc.), while the 4 most significant bits are used to encode the hop count. Fig. 3 depicts an encoding example for steering traffic over the path: $1 \rightarrow 2 \rightarrow 5$ (topology in Fig. 1).

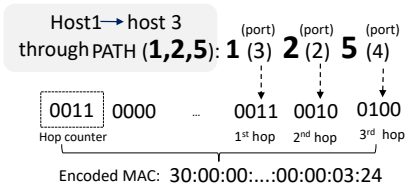

Fig. 3: Source routing MAC header encoding example.

As such, we are in position to encode paths with a maximum of 11 hops, with 16 ports in each bridge, which is deemed sufficient for pure layer-2 operations and controlled environments. In the case of applications requiring a higher number of hops as in the case of industrial automation, this limitation on the hop count can be overcome by resetting the source MAC in each domain by the edge routers of multi-domain TSN. This approach is similar to the one presented in [7]. However, therein the path position is identified based on the TTL value, which requires inspection of the IP header. In our approach, all required operations are applied solely on pure Ethernet frames, ensuring compliance with TSN.

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=416.234s, table=0, n_packets=6, n_bytes=612,
priority=2,dl_dst=00:00:00:00:00:03 actions=set_field:
30:00:00:00:03:24->eth_src,output:2
 cookie=0x0, duration=419.459s, table=0, n_packets=12, n_bytes=868,
priority=0,in_port=1 actions=CONTROLLER:65509
 cookie=0x0, duration=419.458s, table=0, n_packets=57,
n_bytes=6810, priority=0,in_port=2 actions=output:1
```

Fig. 4: Example of flow table for path encoding.

To realize the proposed scheme, between the Talker hosts and the edge TSN switches we add OpenvSwitch (OVS) instances [8] for path encoding. Fig. 4 shows the flow entry used for the source MAC address encoding example depicted in Fig. 3. The *dl_dst* field in the first entry of the flow table is used to perform matching of the packets based on the destination MAC address. The action list applied on the packets that match this entry consists of: (i) replacing the source MAC address with the source routing encoded address, and (ii) forwarding the packets through the indicated output port. The second and third entries in the flow table, shown in Fig. 4, apply the default action taken by the edge switches.

B. Software TSN aware switches

We implement software TSN switch fabric using Time Aware Priority Shaper (TAPRIO) interface inside a virtualized environment. TAPRIO is a Linux QDisc based queuing discipline, which implements 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 (potentially empty) of traffic classes based on the notion of time slice. Furthermore, we integrate TAPRIO into Mininet emulator for the implementation of a TSN-aware network [9]. The required set of changes are applied into Mininet version 2.2 running on Linux kernel version 5.4. Fig. 5 shows the bird's eye view on the implementation of TSN switches in Mininet, indicating the following modifications:

- **Multi-queued NIC interfaces.** TAPRIO is supported only on multi-queued network interface. Hence, in order to configure TAPRIO inside Mininet, we need to create multi-queued network interfaces or Virtual Ethernet pairs inside Mininet instead of single-queue network interfaces.
- **TSN schedules on virtual network interfaces.** Configuration of Linux TAPRIO queuing discipline is carried out via the traffic control tool (TC) [10].
- **Traffic class to VLAN priority mapping.** TAPRIO uses the priority field of the socket buffer used by the networking stack of the Linux kernel (*skb→priority*) to classify the packets to a particular traffic class. As such, the *skb→priority* field needs to be modified based on the VLAN priority field to map the traffic classes of the TAPRIO based on VLAN priority. In order to set this kernel data structure from userspace, we rely on ingress and egress port mapping provided by the VLAN ports. Each OVS datapath is connected to the VLAN interfaces of the virtual Ethernet pairs.

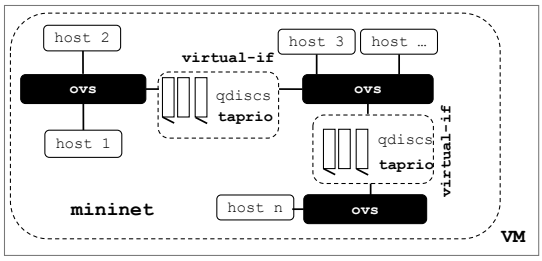

Fig. 5: TAPRIO in Mininet.

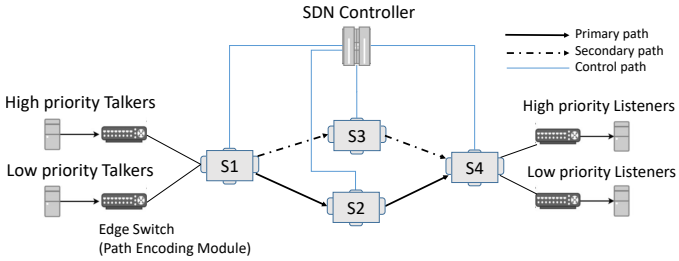

Fig. 7: Experimental network topology

C. Path Decoding

For the forwarding decision at each switch, a local software component needs to be instructed (only once) on the way to decode the path information encapsulated at each packet. The switch extracts the correct port number from the sequence of ports encoded into the source MAC address of the packet header field. In order to perform the decoding operation, we rely on OpenFlow version 1.3 (similarly to the encoding).

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=106.153s, table=0, n_packets=5, n_bytes=490,
priority=1,dl_src=10:00:00:00:00:00/f0:00:00:00:00:00 actions=load:
0->NXM_OF_ETH_SRC[44..47],output:NXM_OF_ETH_SRC[0..3]
 cookie=0x0, duration=106.153s, table=0, n_packets=0, n_bytes=0,
priority=1,dl_src=20:00:00:00:00:00/f0:00:00:00:00:00 actions=load:
0x1->NXM_OF_ETH_SRC[44..47],output:NXM_OF_ETH_SRC[4..7]
......
......
```

Fig. 6: Example of flow table for path decoding.

Fig. 6 shows the flow table entry of each switch serving as the actual packet forwarding node. In our case, the switches extract the corresponding output port from that position in the source MAC address, based on the value in the Most Significant Nibble (MSN) of the MAC address. For example, if the value in the MSN of the source MAC address is 3, the corresponding output port is extracted from the 3rd nibble of the source MAC address, whereas the MSN of the MAC address is decremented by 1.

D. TARPIO Configuration

According to 802.1Qcc, the configuration of the TSN functionality (*e.g.,* GateControlList for scheduled traffic according to 802.1Qbv) can be conducted using a management protocol, such as NETCONF (with respect to the 802.1Qcw YANG models). In this regard, any SDN control platform supporting a NETCONF client can be employed (*e.g.,* OpenDaylight). Since 802.1Qcw YANG models are not yet finalized and a NETCONF API is currently not available for TAPRIO, the TAPRIO configuration is applied using the traffic control tool[1]. The development of a NETCONF interface for TAPRIO is part of our future work.

IV. Performance Evaluation

The main goal of our experiments is to illustrate the impact of source routing and TSN schedules on the performance of the time-critical network applications. For our experiments, we use a simple network topology, consisting of two paths (*i.e.,*, primary and secondary) between the source and the destination pairs, as shown in Fig. 7. The network under test is emulated using TAPRIO enabled Mininet, as explained in Section III. Regarding traffic generation and measurements collection, we rely on Distributed Internet Traffic Generator (D-ITG) [11]. In the following, we present results from three different experiments, showing the performance of TARPIO on a virtual interface (Section IV-A), the scalability of TARPIO while increasing the number of flows (Section IV-B), as well as the combined effect of source routing and TSN on latency (Section IV-C).

A. TAPRIO in Physical and Virtual Interfaces

This experiment aims at measuring the overhead introduced by TAPRIO, when enabled in virtual network interfaces. The corresponding experimental setup is shown in Fig. 8, where the source and destination ports are connected to an external machine, at which the D-ITG traffic generator is deployed. The network traffic generated is tagged ICMP (priority - 7) with a packet size of 100 Bytes at a rate of 2000 packets per second.

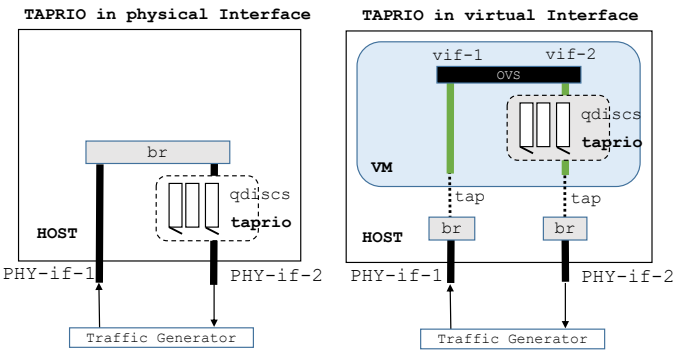

Fig. 8: Experimental setup with physical and virtual TAPRIO.

Fig. 9 illustrates the average latency for different TSN schedules (GCL) with physical and virtual TAPRIO implementations and 1ms cycle time. The plot depicts the performance for high priority traffic, where for following notation X:Y, X corresponds to the duration in μs for which gate for priority 7 traffic is open and Y refers to the duration for which gate for best effort traffic is opened after closing the gate for priority 7

[1]See https://www.frank-durr.de/?p=376 for example configurations,

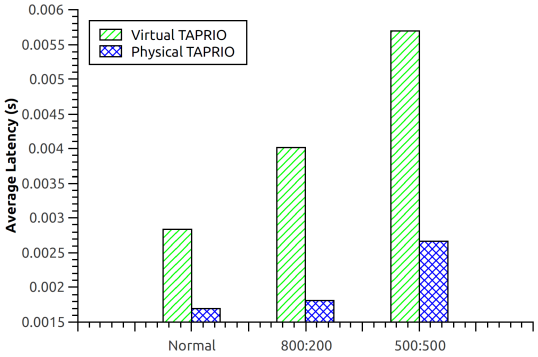

Fig. 9: Average Latency for high priority traffic, comparison between physical and virtual TAPRIO. Notation 800:200 means a GCL entry with 800us for high priority, followed by 200us of best effort traffic for a cycle of 1ms. Normal operation refers to the case where TAPRIO is disabled.

traffic. As expected, the average latency for high-priority traffic decreases monotonously, while increasing the percentage from the overall cycle allocated. Second, the average latency of the virtual TSN switch always exceeds that of its physical counterpart, due to the additional overhead incurred by the transmission of packets through both virtual and physical queuing layer.

B. Scalabilty of TAPRIO

In this experiment, we investigated TARPIO performance while increasing the number of talkers (*i.e.,* sending hosts) and consecutively the number of flows. The experiment is performed on the topology shown in Fig. 7. In particular, we use two different types of traffic generated internally within Mininet: (i) high-priority (Priority 7) tagged ICMP traffic with packet size of 1400 Bytes generated using *D-ITG* at the rate

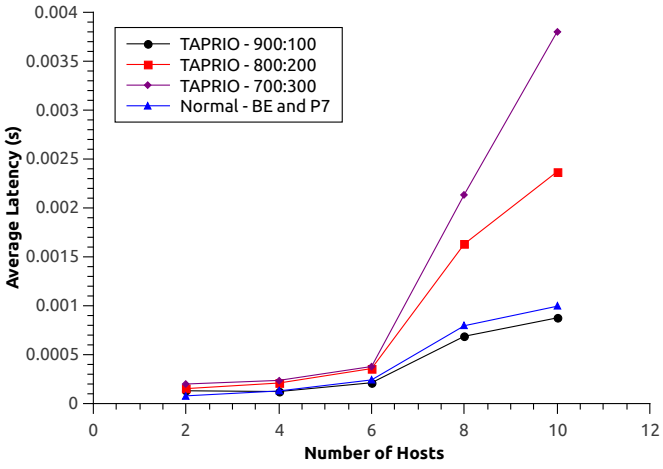

Fig. 10: Average latency performance of high priority traffic, while increasing the number of hosts/flows (X:Y notation similarly to the one used in Fig 9).

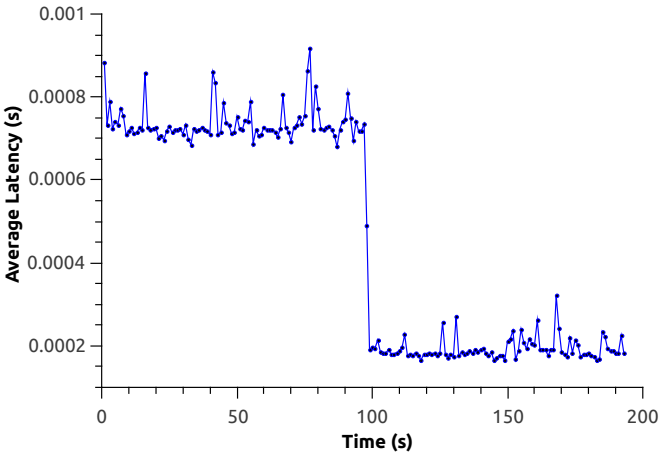

Fig. 11: Impact of best-effort traffic re-routing and rescheduling on the delay of high-priority traffic.

of 5000 packets per second for all talker-listener pairs (with a range of 1 to 10); (ii) best-effort traffic generated using *Iperf*.

Fig. 10 shows the average latency of high-priority traffic, while increasing the number of hosts for different GCL configurations. The TSN schedules in this plot follow the same convention as before. In general, the average latency of the flows increases monotonously with the number of hosts, whereas the variation in latency is more pronounced as well, due to the increased in-class contention on the high-priority queue, while reaching its saturation point. The normal case (*i.e.,* without TAPRIO) yields lower average latency compared to TAPRIO, due to the strict priority scheduling algorithm running by default on the network interfaces. Note that scheduled traffic is typically not used to obtain better average latencies, but deterministic latencies. Nevertheless, TAPRIO offers the flexibility to tune the schedules based on the application's latency requirements. In the case of TAPRIO queue disc, the average latency is increasing with lower values of duty cycle (*i.e.,* the percentage allocated) for high-priority traffic.

C. Impact of Source Routing and TSN Schedules on Latency

This experiment studies the impact of source routing and TSN schedules of best effort traffic on the average latency experienced by high-priority traffic. The experiment is performed using two different types of traffic generated internally within Mininet: (i) high-priority (Priority 7) ICMP traffic with packet size of 1400 Bytes generated using D-ITG at the rate of 10000 packets per second (with constant inter-departure time), flowing through the first talker–listener pair; (ii) ten parallel flows of best-effort interfering traffic with packet size of 1400 Bytes at the rate of 5000 packets per second per flow, generated by a traffic generator application flowing through the second talker–listener pair. The experiment is conducted over a duration of 180 seconds with both the high-priority and best-effort traffic flowing through the network, whereas the initial TSN schedule configuration is set to 50%

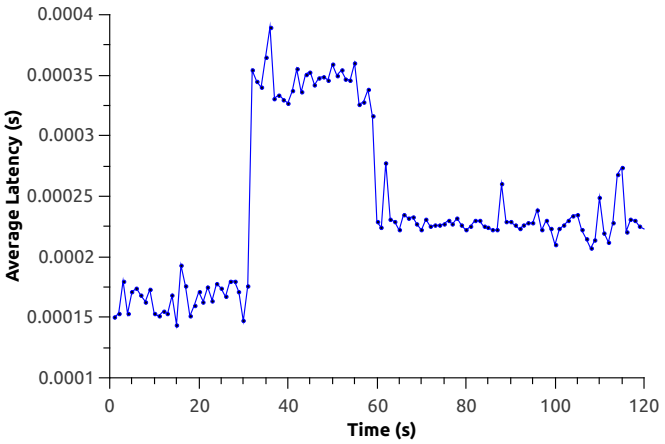

Fig. 12: Impact of re-routing (without rescheduling) on the delay of high-priority traffic.

duty cycle for high-priority traffic. Initially, both best-effort and high-priority traffic are routed along the primary path. After approximately 100 seconds, the SDN controller (*i.e.,* Ryu) re-routes the best-effort traffic to the secondary path and also triggers a scheduling update in the TSN bridges allocating 90% duty cycle for high-priority traffic. Fig. 11 shows the average latency of the high-priority traffic. The average latency experienced by high-priority traffic decreases, due to the combined effect of re-routing of best-effort traffic on the secondary path and increasing the duty cycle for high-priority traffic. We note, however, that source routing alone does not affect the average latency of the high-priority traffic, since this depends on the GCL, cycle and percentage of the cycles allocated.

In the following experiment, we investigate the effect of re-routing the high-priority interfering traffic. The experiment is performed by replacing ten flows of best-effort interfering traffic with high-priority traffic on the second talker–listener pair. The TSN bridges are configured with 70% duty cycle for high-priority traffic. The duration of the experiment is 180 seconds. Initially, there is only a single high-priority flow from a single talker to a single listener. After 30 seconds, interfering background traffic is injected in the network, while ten additional high-priority flows are routed along the primary path. After approximately 60 seconds, Ryu re-routes the additional interfering high-priority traffic to the secondary path.

Fig. 12 shows the average latency of the high-priority traffic flowing through first talker–listener pair. The increase in the average latency at approximately 30 seconds is due to the addition of the high-priority interfering traffic flowing through the primary path (*i.e.,* in-class interference). At approximately 60 seconds, the average latency decreases, due to the re-routing of interfering traffic to the secondary path, which reduces the interference on the primary path. Nevertheless, the average latency is higher than the one measured over the first 30 seconds, due to the in-class interference, since the bridges 1 and 4 are used by both primary and secondary paths.

V. Conclusions and Future Work

In this paper, we coupled source routing with TSN in order to create explicit paths, facilitate re-routing, and reduce the forwarding state in switches. We presented an implementation in Mininet exploiting the TAPRIO interface, together with SDN control and Ethernet frame header encoding. In our experimentation, we quantified the gains of re-routing interfering (best-effort) traffic on the latency of high-priority traffic. We further investigated the effect of combined re-routing and rescheduling (*i.e.,* increase of duty cycle), harnessing both source routing and TSN. In essence, the conjoint use of source routing and TSN schedules comprises an enabler for the development of optimal scheduling policies in order to satisfy the strict requirements of network services at low cost. Note that the simple encoding/decoding mechanism proposed can be easily integrated in devices with TSN compliant hardware, as long as there is SDN (e.g., Openflow) support.

In our future work, we will conduct a thorough investigation of the system and statistical parameters that affect the joint source-routing and TSN scheduling performance, while also seek to develop a relevant joint optimization framework. We note that typically scheduled traffic is not used to obtain better average but deterministic latency. We further plan to present a statistical analysis of delay and jitter, while also study the effects of our approach on throughput performance. In addition, we will investigate techniques that overcome the 11-hops limitation in the proposed encoding, meeting the requirements of networks with a large diameter.

References

[1] 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 Communications Surveys Tutorials*, vol. 21, no. 1, pp. 88–145, 2019.

[2] N. G. Nayak, F. Duerr, and K. Rothermel, "Routing algorithms for ieee802.1qbv networks," *ACM SIGBED Review*, vol. 15, no. 3, pp. 13–18, 2018.

[3] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (tssdn) for real-time applications," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 193–202.

[4] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in ieee 802.1qbv time sensitive networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 183–192. [Online]. Available: https://doi.org/10.1145/2997465.2997470

[5] S. A. Jyothi, M. Dong, and P. B. Godfrey, "Towards a flexible data center fabric with source routing," in *ACM SIGCOMM*, 2015, pp. 1–8.

[6] C. Papagianni, P. Papadimitriou, and J. S. Baras, "Towards reduced-state service chaining with source routing," in *IEEE/ACM CNSM*, Nov 2018, pp. 438–443.

[7] K. Papadopoulos and P. Papadimitriou, "Leveraging on source routing for scalability and robustness in datacenters," in *IEEE 5GWF*, 2019, pp. 148–153.

[8] B. Pfaff and B. Davie, "The open vswitch database management protocol," *IETF RFC 7047*, 2013.

[9] *MIninet*. [Online]. Available: http://mininet.org

[10] B. Hubert *et al.*, "Linux advanced routing & traffic control howto," *Netherlabs BV*, vol. 1, 2002.

[11] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.