Received 21 June 2023, accepted 1 July 2023, date of publication 7 July 2023, date of current version 18 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3293061

RESEARCH ARTICLE

DTPF Algorithm Based Open-Source Time-Sensitive Network Leveraging SDN Architecture

ASHA G. HAGARGUND[1,2], (Graduate Student Member, IEEE), NEELAWAR SHEKAR VITTAL SHET[1], AND MURALIDHAR KULKARNI[3], (Senior Member, IEEE)

[1]Department of Electronics and Communication Engineering, National Institute of Technology Karnataka, Surathkal 575025, India
[2]Department of Electronics and Communication, BMS Institute of Technology and Management, Bengaluru 560064, India
[3]Department of Electronics and Communication Engineering, National Institute of Technology Karnataka, Surathkal 575025, India *(Retired)*

Corresponding author: Asha G. Hagargund (asha.187ec500@nitk.edu.in)

ABSTRACT Time-Sensitive Networking (TSN) has enabled a lot of advancements in industrial automation, aviation, tactile networking, and other ad-hoc networking applications. The bounded latency, reliability, and self-recovering mechanisms for a network are some of the core attributes of the TSN architecture. The reliability of bench-marking of a given TSN architecture, when done with simulators, will not match with that of hardware systems. But the TSN hardware needs more capital investment along with more development time as it involves understanding the hardware-specific parameters. The research in this paper has solved this shortcoming by implementing an open-source and secured SDN(Software Defined Networking)-based TSN framework integrating IEEE's 802.1 Qbv and 802.1Qcc standards. The novelty of implementation involves i) Realization of Centralized User Configuration (CUC), Centralized Network Configuration (CNC) with open source tools. ii) An algorithm called Dynamic TSN Path Finder (DTPF) is implemented for automatically identifying TSN edges participating in TSN flow. iii) Emulation of hardware environment leveraging Linux-based queuing disciplines and traffic shapers. This unique open source-based TSN architecture is then tested with both TSN and Non-TSN traffic, to demonstrate the gating logic and the delay characteristics based on queuing discipline when applied to virtual queues.

INDEX TERMS Software-defined networking, open-source, scheduling, switches, queuing disciplines, time-sensitive networking, IEEE protocols.

I. INTRODUCTION

TSN is an effort to integrate Operational Technology (OT) and Information Technology(IT) enabling the co-existence of time-bounded control traffic and best-effort traffic to reduce the infrastructure cost. The IEEE 802.1 TSN standards provide real-time capabilities to the Ethernet, which have opened up a large number of applications in Industrial Automation, Audio Video Bridging (AVB), Aerospace, Automotive, and in-vehicle communication domains [1]. These standards aim at providing deterministic data delivery with bounded latency for heterogeneous data and shared networks creating a vendor-independent and converged network.

In current automotive and industrial networks, the reconfiguration of the network is not possible on the fly whenever there is a need. This evidently affects the efficiency of the network as it is more time-consuming because of manual effort, which is vulnerable to human errors [5]. Hence there is a need for a centralized controller which can perform automatic scheduling in case of network reconfiguration. [4], [17], [35], [39]

The SDN architecture has a centralized controller that controls flow tables of switches connected to it. Hence leveraging SDN architecture for TSN brings down the development time significantly [22], [24], [39]. Most of the papers propose

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

customization to the existing SDN framework that is required for realizing TSN architecture. The paper [5] proposes one such SDN-based TSN model using NEON controller and APIs required for communication with TSN bridges which are compliant with IEEE's 802.1Qbv and 802.1AS standards.

Similarly, the author in [7] proposes and implements the TSN network architecture for industrial automation especially for dynamically interconnected time-sensitive end devices, using SDN-based centralized configuration and Open Platform Communications-Unified Architecture (OPC-UA) which serve as the data exchanging technology between applications of industrial automation.

The author in [12] has created an SDN-based TSN network for industrial Ethernet applications with multiple Quality of Service (QoS) requirements. Depending on the application's QoS requirements the appropriate TSN schedules are pushed from the SDN-based controller to its connected switches. The scheduling strategies are calculated at the SDN controller and are used at the switches to update their Gate Control List(GCL). The communication between the SDN controller and the TSN switches is done through the NETCONF protocol. The Satisfiability Modulo Theories (SMT) based scheduling and rate monotonic scheduling have been realized.

In the paper [14], the author proposes the partitioning of available bandwidth for implementing IEEE 802.1Qbv scheduling. This approach enables the SDN controller to calculate the configuration of dynamic flows independently without disturbing the static configuration. The simulation for finding the end-to-end delay and jitter for various in-car network nodes is carried out in an OMNET++ based simulator. Hence the results presented do not take real-time constraints into consideration.

The paper [15] implements an SDN-based TSN network using source routing and IEEE 802.1Qbv in its data plane to check the performance of the network for bounded latency using Linux TAPRIO(Time Aware Priority Queuing) queuing discipline. The virtual local area network's Virtual LAN(VLAN) Priority Code Point (PCP) based traffic class mapping is used with the multiple virtual queues. The research work under [15] does not give much detail on how virtual queues are created and also does not emphasize the classification of packets required for queuing discipline. The source-based routing is proposed in [15] without discussing the overhead imposed on TSN traffic.

The author of the paper [19] proposed a reconfiguration framework based on the IEEE 802.1Qbv standard in collaboration with the IEEE 802.1Qcc standard using SDN controller to implement a control mechanism that incorporates the 802.1Qcc variables, including the flow instantiating parameters and the 802.1Qbv gate control parameters. The author modeled his own queue structure for the simulation of aforesaid IEEE standards. The paper [19] does not convey the type of SDN controller used. The information given as part of virtual queues is not justifiable with respect to gate control logic that has been proposed by IEEE 802.1Qbv standards. Also, the Author in [18]explains the SDN model, to implement the inter-domain TSN network, and the author in [20] builds a test bed to demonstrate the unified control-plane Time-Sensitive Software-Defined Networking (TSSDN).

The fully centralized architecture inspired by the SDN model of TSN is proposed by IEEE [25]. According to this CUC is a software that considers the requirements from TSN end nodes and sends the requirement to CNC. The CUC collects the requirements of TSN end devices through CUC and sends this information to TSN switches. TSN switches are capable of transmitting and receiving the Ethernet frames according to the schedule. TSN talkers and listeners are the source and destination of the TSN flows. This model has knowledge of the entire network and hence manages the transmission schedule and paths.

In [44], the author discusses about the issues in integrating the SDN control plane for TSN. Implements a small-scale hardware test bed using NXP LS1028A to build CUC, CNC, and TSN switches and NXP iMX8MP as TSN end nodes. The delay for TSN data is evaluated. As SDN-based architecture might produce high latency when network reconfiguration is done in the control plane, the author concludes that the proposed architecture, Fully time-sensitive software-defined networking helps in optimizing the problem.

Creating a complete hardware setup is a costly and complex task, especially for a large-scale TSN network. The author in [46] implements the hybrid simulation testbed, where software-generated traffic is combined with real sensor data. With this setup, they test for security, data processing, protocol testing, and performance measurement. The hardware components used are Ubertooth One and Libelium Waspmote for generating sensor data, C1-Toradex as SDN controller, and for software Mininet and Mininet-Wifi in a Linux environment are used.

In [33], the author proposes and implements open-source scheduling strategies for Linux systems. The new TT(Time Triggered) scheduler achieves real-time guarantees. The details on various scheduling mechanisms for real-time applications are mentioned. This paper doesn't consider SDN-based architecture.

As discussed in the previous paragraphs, many papers are present in the literature which implements and tests TSN network with simulation tools [14], hardware test beds [41], [44], [45] and hybrid of both hardware and software [46]. As we have used virtual queues for our work, it is important to discuss the virtual queues and compare them with hardware queues.

The hardware queues are part of NICs that enable offloading of network processing tasks from the host's CPU. Hence this reduced CPU utilization helps in boosting networking performance. Whereas the virtual queues are implemented in software. The networking stack of the host operating system takes the full onus of handling traffic passing through virtual queues. The virtual queues are typically used for communication between different network namespaces or containers within a host. Hence, the CPU speed of the host, the networking stack efficiency, and the system load are some of the factors impacting the service rate of the virtual queue. This might lead to increased latency and lower throughput compared to the hardware queue. Intel®Ethernet Controller I210 (Network Interface Card)NIC from Intel is one of the TSN-aware NIC with hardware-based queues. These kinds of hardware-based NICs provide bounded latency communication for industrial applications. [41].

The continuous evolution of the kernel bypass techniques and software data planes have greatly succeeded to address the performance limitations of traditional software-based networking stacks, narrowing the performance gap between virtual queues and hardware queues. Intel's Data Plane Development Kit (DPDK) enables user-space applications to achieve high packet processing rates by bypassing kernel [42].

In our paper, we have shown how software containers and their virtual queues can be used to provide deterministic operations suitable for industrial networks. On the same lines the research proposed under [41] demonstrates the accuracy of software application timing and offset control is within a few hundred- μsecond range, which can satisfy the requirements posed in the cloud environment.

Our research work proposed in this work emphasizes on using our architecture for TSN-aware PoC(Proof of Concept) as our novel architecture uses open-source tools to bring down the PoC cost.

IEEE proposes scheduling strategies like IEEE 802.1Qbv (Time Aware Shaper), IEEE 802.1Qbu(Frame Preemption) and IEEE 802.1Qav(Credit Based Shapers). In our work, we assume that TSN traffic is periodic and the TSN traffic arrival time is known prior. For this requirement, IEEE 802.1Qbv is the apt scheduling scheme. TAS ensures that the egress port which is designated for TSN traffic is idle when TSN traffic is scheduled. It ensures this by using the gating logic. TAS assigns the queues to each traffic class and the gates to their allocated queues are kept open or close based on the schedule. As in periodic TSN data transmission, the schedule is known prior to the gate corresponding to the TSN traffic class being kept open, so that a particular queue gets attached to the egress port. This makes sure that TSN traffic egresses at the scheduled time [31], [43], [47], [48].

Since our work mainly focuses on TSN traffic where the traffic schedule is already known to the admin, we have decided to do the research using IEEE802.1Qbv. Moreover, we wanted to explore the performance of the veth queue without pushing ourselves into the number of queue limitations.

IEEE 802.1Qcc proposes 3 architectures. Fully distributed, fully centralized and hybrid architecture [44]. We use fully centralized architecture as it proposes centralized management and control which enables seamless pushing of policies to all the connected edges. We have leveraged the SDN model to implement a fully centralized architecture which helps in pushing TSN schedules to all the switches participating in a TSN path/flow.

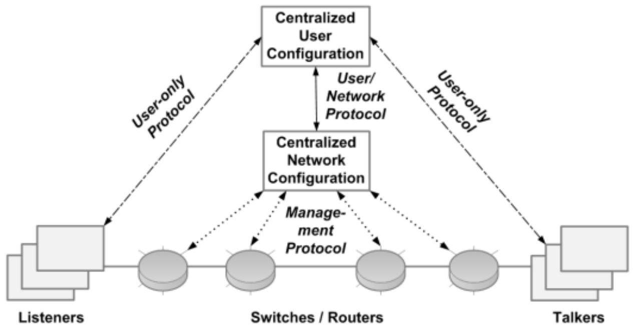


FIGURE 1. **IEEE 802.1Qcc Fully centralized architecture [3].**

The paper is structured in various sections as mentioned here. Section II- Describes the existing research gaps and the objectives of our paper. Section III describes our design approach carried out for the implementation of open source secured and SDN-based TSN architecture and section IV explains our contribution. SectionV discusses the results and conclusion.

II. RESEARCH GAP AND OBJECTIVES

A. RESEARCH GAP

In literature, many papers discuss about the need for a centralized SDN structure for TSN. There is a need for taking the real-time traffic and real constraints into consideration for bench-marking the results.

When SDN architecture is implemented for IEEE 802.1Qbv, using Linux queuing disciplines, configuration related to multiple virtual queues needs to be discussed with related details. These are missing in the present literature.

TAPRIO queuing discipline demands traffic classification at ingress and egress queues for allocation of queues. But the papers mentioned in the introduction do not present configuration and implementation details. Most of the papers modify OpenvSwitch to incorporate QoS functions needed for TSN. This may lead to some errors as OpenvSwitch may not modify every Linux kernel feature due to non-registered bugs if any [29]. To address the above-mentioned gaps the following objectives are defined in section II-B.

B. OBJECTIVES

- Perform the validation of the IEEE 802.1Qbv gating strategy by considering the real-time traffic.
- Provide complete implementation details of multiple virtual queues for the egress port of the switch.
- Discuss all the configuration details on TAPRIO, traffic class, filtering, and traffic scheduling in detail.
- Instead of modifying the OpenvSwitch for incorporating QoS functions, use the Linux traffic control utility "tc".
- Implement and validate the algorithm DTPF for TSN traffic for routing.
- Make communication secure by using Paramiko.

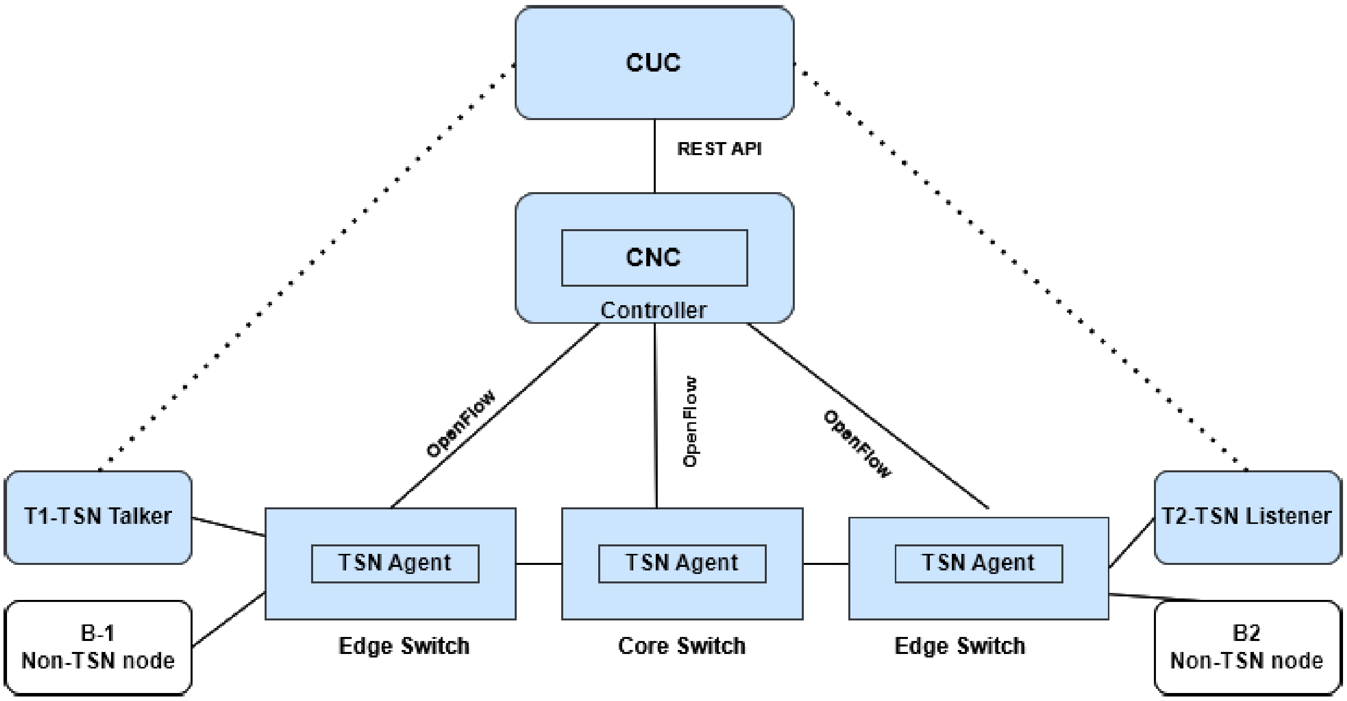


FIGURE 2. Basic building blocks of an SDN-based TSN network based on IEEE 802.1Qcc.

III. WORK DONE

A. DESIGN APPROACH

We propose a TSN network, that is based on IEEE 802.1Qcc fully centralized network as depicted in figure 1.

According to this CUC is software that communicates with the end devices and CNC. CUC gets the requirements of the end devices and makes requests to the CNC for deterministic communication with specific requirements for the traffic. Whereas CNC defines the schedule on which all TSN frames are transmitted. The CNC application is provided by the vendor of the TSN bridges. By taking this as a reference, we build the software for CNC, CUC, and TSN agents. We consider 2 traffic classes, one for TSN traffic and the second one for Non-TSN(Best Effort) traffic. We have 2 source nodes and 2 corresponding destination nodes as shown in fig.2. The following subsection explains the functions and implementation details of each component mentioned in Fig.2.

1) CENTRALIZED USER CONFIGURATION (CUC)

The CUC is a vendor-specific implementation for associated TSN nodes [26], [28]. This is an independent device provided by the vendor. In our implementation, CUC is hosted as a docker container. Under this container, a program called CUC-agent communicates with the Talker, Listener, and the CNC over a secured channel provided by SSH [2]. CUC is responsible for generating the TSN flow by pulling the requirements from TSN end nodes. CUC will then query the topological information from CNC using REST API. The topological information is filtered to form TSN flow, using our algorithm DTPF. Eventually, this TSN flow is sent to CNC. A TSN flow is a unidirectional path from talker to listener including the intermediate nodes as clearly depicted in Fig. 2

2) CENTRALIZED NETWORK CONFIGURATION (CNC)

We have used the OpenDaylight controller as CNC. But, we have not customized any OpenFlow API's for TSN architecture. Instead, a program called CNC-agent discovers the entire topology and returns it to the CUC when queried as mentioned in subsection III-A1 . On receiving the flow requirements from the CUC, the CNC will generate the TSN schedule according to IEEE 802.1Qbv standard. A TSN schedule contains the configuration which specifies the allocation of priority, queue, and time slice for a particular or set of traffic classes.

3) TSN ENABLED SWITCH

This is an SDN's OpenvSwitch in our implementation. To make this switch TSN aware a program called TSN agent is executed on the switch which deploys the TSN schedules as received from the CNC.

4) TALKER AND LISTENER

In our implementation, these are the two docker containers. The TSN end devices are responsible for generating and receiving the time-sensitive data respectively. The greater details on the classification of TSN data traffic are given in section IV-C

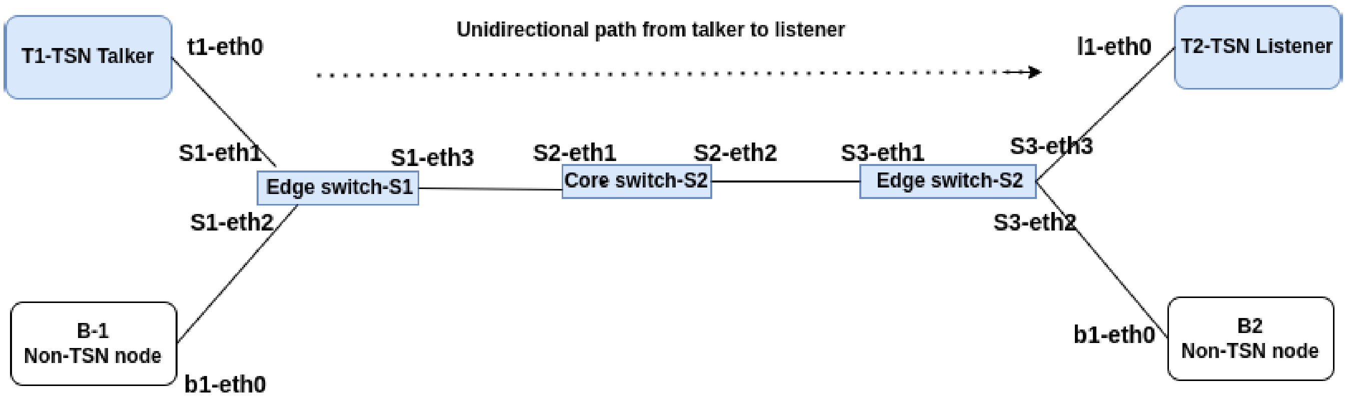


FIGURE 3. TSN flow/ TSN path with ethernet interfaces.

TABLE 1. Tools used for experiment.

Name of the component	Version
Mininet	2.3.1.b1
OpenDaylight controller	3.0.3
OpenVswitch	2.17.5
Python Programming language	3.8.10
Ubuntu O.S	20.04
"tc" utility	iproute 2 22200127

1) Talker's MAC and IP
2) Time slice of TSN data.
3) Listener's MAC and IP.

As mentioned in the objectives, we use secured communication between the components in our network by using SSHv2(Secure Shell Protocol Vesrion 2) Scripting protocol. The Python program uses paramiko, a module that implements the SSHv2 protocol for authenticating to a server using a password and SSH keys. Only after successful authentication, further data communication is done over a secured channel.

B. TOOLS AND COMPONENTS

Table 1 lists the information on open-source tools and other Linux utilities used in this work.

IV. OUR CONTRIBUTION

A. DYNAMIC TSN PATH FINDER (DTPF)ALGORITHM

For configuring the TSN schedule on edges and core switches involved in a TSN flow, the TSN path has to be identified first. To identify the TSN path we have implemented an algorithm called Dynamic TSN Path Finder which extracts the nodes that are part of the TSN flow. The input data for this algorithm is the topological information obtained from CNC using REST API by CUC. The proposed algorithm is mentioned in the algorithm section.

B. NOT CUSTOMIZING SDN AND OPEN vSwitch MODELS FOR TSN

The research in this paper does not propose customization of SDN controllers as opposed to the paper discussed in the introduction section. Typically SDN controller customization involves writing/rewriting SDN flow APIs as per the requirement of the TSN network. These APIs in turn communicate to OpenvSwitch for updating its flowtable. This involves the configuration of a subset of Linux kernel QoS features. OpenvSwitch forum claims that it cannot configure every Linux kernel QoS feature because of unreported bugs related to QoS if any [29]. To avoid any side effects of unknown bugs we have configured the QoS using Linux traffic control utility "tc". [21], [30]. The SDN concept was proposed by Open Networking Foundation (ONF) that decouples the control and data planes. The framework is not directly usable for TSN architecture, as TSN demands core Linux QoS parameters modifications. If the underlying architecture of TSN is SDN, then all the challenges that are applicable to SDN also hold good for TSN. The main requirement in the TSN network is to have bounded latency between TSN talkers and listeners. Hence this paper does not propose a modification to the SDN controller model.

Algorithm 1 Dynamic TSN Path Finder Algorithm

```
Input: $srcmac, dstmac, sip, dip, ms$
Output: $tsnpath$
1: Get edges connections info from CNC using REST API
2: Get talkerswitch using smac
3: Get listenerswitch using dmac
4: prevswitch = none
5: get nxtswitch (switch connected to Talker switch)
6: curswitch (Talker switch)
7: tsnpath += nxtswitch
7: While $nxtswitch \neq listenerswitch$ do
8: prevswitch = curswitch
9: curswitch = nxtswitch
10: nxtswitch = Get the switch connected to nxtswitch
11: tsnpath += nxtswitch
11: EndWhile
12: Tsnpath has all intermediate TSN edges constituting TSNpath
```

TABLE 2. Commands to classify ingress traffic.

```
sudo tc qdisc add dev s1-eth1 clsact
sudo tc filter add dev s1-eth1 ingress prio 1 u32 match ip dst
10.0.0.3 action skbedit priority 1
sudo tc filter add dev s1-eth1 ingress prio 1 u32 match ip dst
10.0.0.4 action skbedit priority 0
```

C. TSN TRAFFIC CLASSIFICATION

Fig. 3 shows the topology with egress and ingress port details of TSN edge and core switches. As we already discussed in section II-B the TSN traffic is not controlled using Open-vSwitch internal commands. Instead, we use a native Linux tool called "tc". TSN-based flows involve multiple queues controlling egress traffic based on various classes of traffic. But, Mininet by default creates virtual ethernet links with a single transmit and receive queue. To enable multiple queues under TSN aware switch we have modified the veth configuration under mininet version 2.3.1b1 to increase the number of transmit queues to 8. [30]

1) INGRESS TRAFFIC CLASSIFICATION

The ingressing traffic on the s1-eth1 port should be classified as TSN traffic. This also implies that the TSN traffic should be handled with high priority. To achieve this, Linux queuing discipline called "clsact" is used. The queuing discipline "clsact" enables the user to classify the packets based on the direction of the traffic on a given port. The classification can be made based on parameters like port, IP, and MAC [32]. The priority assigned to such classified traffic maps internally to socket buffers data structure "skb" priority member [4]. The skb->priority will be used by the Linux scheduler to dispatch the packets for processing based on the priority value. The higher value of skb->priority depicts more priority and is handled first. Table 2 shows the detailed "clact" commands used for classifying the ingressing TSN traffic on a port.

The command "tc" is a Linux utility for controlling the traffic. Under this utility, the user is able to provide various queuing disciplines as per the requirement. We use TAPRIO queuing discipline to implement IEEE 802.1Qbv scheduling.

Fig.6 and Fig. 7 shows the results of traffic filtering done using "skbprio" at the ingress port of s1 for TSN and Non TSN data.

D. TAPRIO QUEUING DISCIPLINE

After classifying the ingressing TSN traffic, we need to apply a queuing discipline on the egress port of the switch which should dequeue the prioritized packets at the ingress. To do this we have used Linux queuing discipline called TAPRIO, which is a simplified implementation of scheduling state machines as defined by IEEE 802.1Q-2018 Section 8.6.9. Each gate corresponds to a queue that dispatches outgoing traffic based on the traffic classes. TAPRIO operates on traffic mapped to different hardware queues [23]. We have used "veth" based virtual queues as we are using a native Ubuntu operating system running on Intel(R) Core(TM) i5-8265U

TABLE 3. Command to add virtual Ethernet interface.

```
ip link add name %s numtxqueues 8 type veth peer name %s
```

TABLE 4. Commands to execute TAPRIO on the egress port.

```
sudo tc qdisc add dev s1-eth3 parent root handle 100: taprio
num_tc 2
map 1 0
queues 1@0 1@1
base-time 0
sched-entry S 01 300000
sched-entry S 02 700000
clockid CLOCK_TAI
```

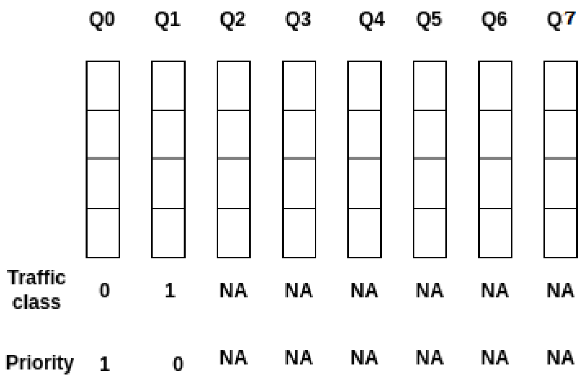


FIGURE 4. Traffic classes and priorities at the egress port of each switch.

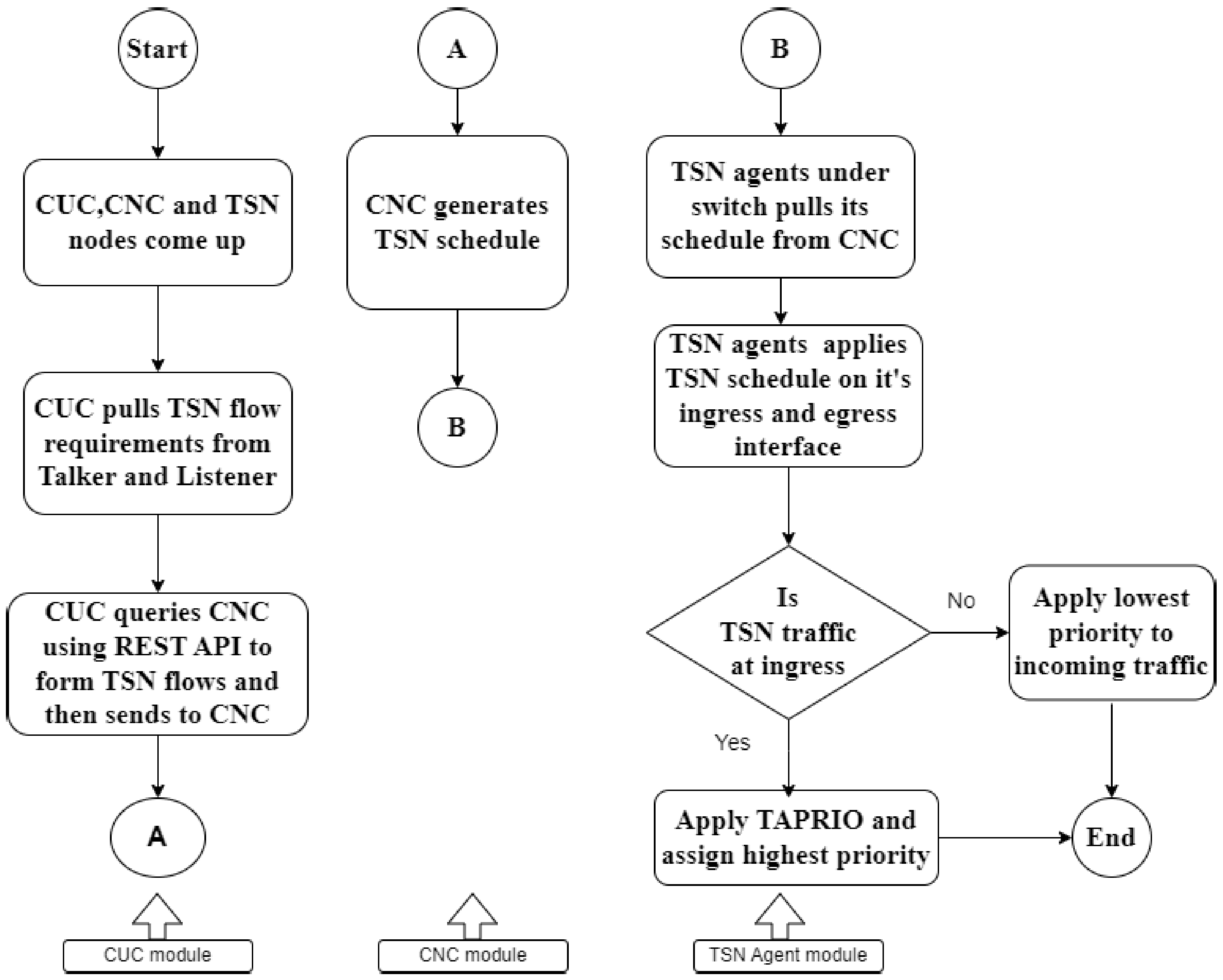
CPU @ 1.60GHz which has a single hardware transmit and receive queue. For configuring virtual queues on Mininet's OpenVswitch interface we have edited the util.py of Mininet. This modification adds 8 transmit queues to the veth interface of OpenvSwitch. The modified version of the command to add the veth interface is given in Table 3 for reference.

Where "%s" is the OpenVswitch interface. For example, "s1-eth1" is the eth1 interface of switch-S1. The "numtxqueues" specifies the number of transmit queues for each interface type.

Table 4 gives a detailed TAPRIO command executed on the egress port [34]. In our examples, 2 traffic classes are used. As shown in Figure 4 TSN traffic is class 0 and Best effort is class 1. Traffic class 0 has priority 1 and traffic class 1 has priority 0. Traffic class 0 uses the 0th queue and traffic class 1 uses the first queue. We expect the TSN traffic at the 0th queue and the best-effort traffic at the first queue. As TSN traffic considered here is periodic, the time slice for TSN traffic is 300 μs and for best-effort, it is 700 μs out of a 1000 μs switch cycle. Fig. 9 shows the results on TAPRIO queuing discipline implementation.

E. SOFTWARE MODULES

When the topology comes up various software modules interact with each other for handling the TSN traffic as per the configuration. The relevant software modules are implemented for carrying out the operations responsible for the



Start
CUC,CNC and TSN nodes come up
CUC pulls TSN flow requirements from Talker and Listener
CUC queries CNC using REST API to form TSN flows and then sends to CNC
A
CUC module
A
CNC generates TSN schedule
B
CNC module
B
TSN agents under switch pulls its schedule from CNC
TSN agents applies TSN schedule on it's ingress and egress interface
Is TSN traffic at ingress
No
Apply lowest priority to incoming traffic
Yes
Apply TAPRIO and assign highest priority
End
TSN Agent module

FIGURE 5. Software module interaction.

creation of TSN flow and generation of TSN schedules for TSN-aware switches. The flowchart presented in figure 5 gives the sequence of events involving below software modules.

1) CUC.py -THIS MODULE IS RESPONSIBLE FOR THE FOLLOWING TASKS

1) Pulling out TSN flow requirements from talker and listener.
2) Creating a TSN flow using the REST API involved DTPF.
3) Sending TSN flow to CNC.

2) CNC.py - THE CNC IN OUR IMPLEMENTATION IS A NON-CUSTOMIZED OpenDaylight CONTROLLER

Since the SDN-based controller has already well-defined control operations using OpenFlow protocol, our goal is to not overload with TSN-related operations. Hence, the CNC.py just parses the TSN flow as received by CUC and generates TSN schedules for switches involved in the TSN path using native Linux QoS commands.

3) TSNagent.PY

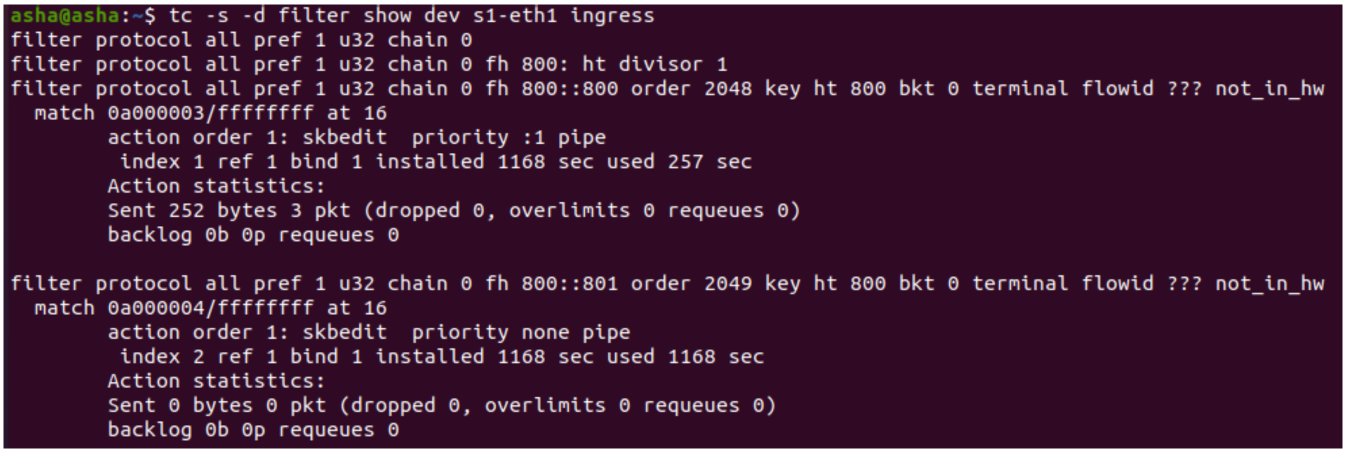
The OpenvSwitch is made TSN aware by adding the TSNagent.py module. We emphasize the point that there is no customization of OpenvSwitch. The TSNagent.py will pull the TSN schedule from CNC and execute the commands present in the schedule on both the ingress and egress interfaces of the TSN switch.

V. RESULTS

Following subsections explain various results observed with our proposed TSN architecture.

A. PACKET CLASSIFICATION USING SKB PRIORITY

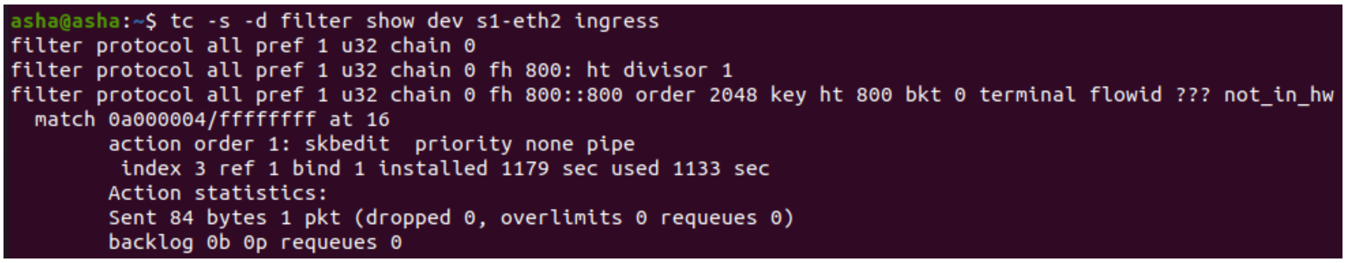
The commands output in Fig.6 and Fig.7 shows the matching criteria for classification of the TSN and BE packets respectively. We are classifying packets based on destination



```
asha@asha:~$ tc -s -d filter show dev s1-eth1 ingress
filter protocol all pref 1 u32 chain 0
filter protocol all pref 1 u32 chain 0 fh 800: ht divisor 1
filter protocol all pref 1 u32 chain 0 fh 800::800 order 2048 key ht 800 bkt 0 terminal flowid ??? not_in_hw
  match 0a000003/ffffffff at 16
        action order 1: skbedit  priority :1 pipe
         index 1 ref 1 bind 1 installed 1168 sec used 257 sec
        Action statistics:
        Sent 252 bytes 3 pkt (dropped 0, overlimits 0 requeues 0)
        backlog 0b 0p requeues 0

filter protocol all pref 1 u32 chain 0 fh 800::801 order 2049 key ht 800 bkt 0 terminal flowid ??? not_in_hw
  match 0a000004/ffffffff at 16
        action order 1: skbedit  priority none pipe
         index 2 ref 1 bind 1 installed 1168 sec used 1168 sec
        Action statistics:
        Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
        backlog 0b 0p requeues 0
```

FIGURE 6. Results on packet classification at s1-eth1 using skb priority.



```
asha@asha:~$ tc -s -d filter show dev s1-eth2 ingress
filter protocol all pref 1 u32 chain 0
filter protocol all pref 1 u32 chain 0 fh 800: ht divisor 1
filter protocol all pref 1 u32 chain 0 fh 800::800 order 2048 key ht 800 bkt 0 terminal flowid ??? not_in_hw
  match 0a000004/ffffffff at 16
        action order 1: skbedit  priority none pipe
         index 3 ref 1 bind 1 installed 1179 sec used 1133 sec
        Action statistics:
        Sent 84 bytes 1 pkt (dropped 0, overlimits 0 requeues 0)
        backlog 0b 0p requeues 0
```

FIGURE 7. Results on packet classification at s1-eth2 using skb priority.

IP address. The destination IP addresses "0a000003" and "0a000004" are of listener and peer BE nodes respectively. The destination IP address if matched to that of listener node, then the "skb" priority of "1" is assigned to that packet. Whereas the destination IP address if matched to that of peer BE node, then the "skb" priority of "0" is assigned to that packet. This assignment will stay intact till the TAPRIO queueing discipline is applied on the packet at the egressing interface "s1-eth3" of switch "s1" Point to note that "s1-eth1" and "s1-eth2" are ingress interfaces on TSN switch "s1" connected to TSN and BE nodes respectively.

B. GCL CHARACTERISTICS

In this section, the results of the realization of gating logic according to IEEE 802.1Qbv is discussed. Fig.8 and 9 depicts the gating logic involving time window allocation for TSN and Best Effort traffic respectively. As per the queuing logic, the first 300 μs of the duty cycle is for TSN traffic and the remaining 700 μs of the duty cycle is for Best-Effort traffic. The observed average time windows during which the TSN queue(Q0) was closed and opened were 734.8 μs and 281.4 μs respectively. This pattern is repeated for every switch cycle of 1000 μs.

Simulation is carried out for 10 ms and packets are generated at the rate of 1 packet per μs using ping utility. The graph in the Fig.8 and Fig.9 depicts that from 0-300 μs of the switching cycle TSN traffic of 1 packet per μs is egressing

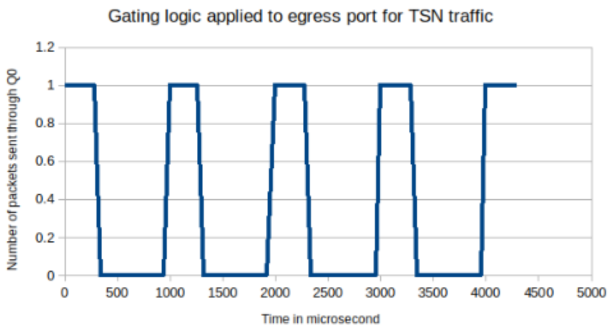


FIGURE 8. Demonstration of gating logic for TSN traffic.

through its respective queue. And from 301-1000 μs the TSN traffic is not allowed through the egress port. This is repeated for each switch cycle of 1000 μs.

The command output in Fig.10 shows the details on TAPRIO queueing discipline applied on interface "s1-eth3". The numbers 3000000 and 7000000 represent the time windows of TSN and BE traffic in nanoseconds, constituting the duty cycle of 10000000 nanoseconds or 1000 seconsds. The gatemasks 0×1 and 0×2 represent the TSN and BE traffic getting mapped to queue 0 and 1 respectively.

C. DELAY CHARACTERISTICS

In this section, various types of delays involved in transmitting TSN data from the TSN talker to the TSN listener are

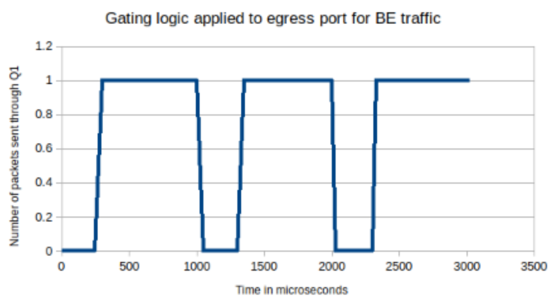
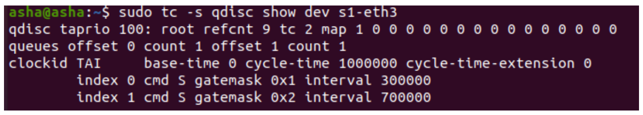


FIGURE 9. **Demonstration of gating logic for Best-effort traffic.**



```
asha@asha:~$ sudo tc -s qdisc show dev s1-eth3
qdisc taprio 100: root refcnt 9 tc 2 map 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
queues offset 0 count 1 offset 1 count 1
clockid TAI     base-time 0 cycle-time 1000000 cycle-time-extension 0
        index 0 cmd S gatemask 0x1 interval 300000
        index 1 cmd S gatemask 0x2 interval 700000
```

FIGURE 10. **Results on TAPRIO queuing discipline.**

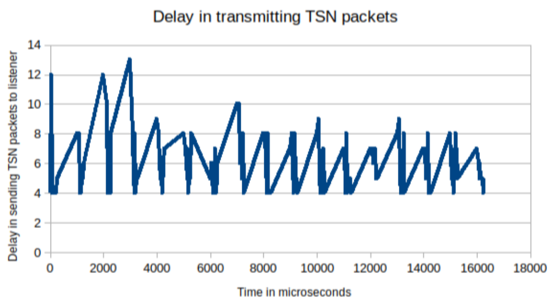


FIGURE 11. **Delay in transmitting TSN packets from talker to listener.**

discussed. The delay experienced at the egress interface for any switch is given by

$$D = Dpr + Dqu + Dtr + Dpg. \tag{1}$$

In Eq.(1) Dpr is the processing delay, which is the time taken for a packet to get processed before queuing, and Dqu is the queuing delay, which is the time that a packet spends in a queue while waiting for other packets to be transmitted, Dtr is the transmission delay, which is the time required to put a whole packet on to the transmission medium and Dpg is the propagation time, which is the time taken by the packet using the link to reach the destination.

We emphasize that there is no control over processing delay as processing delay is completely application dependent. The transmission delay is directly proportional to the processing delay and Queuing delay. By selecting the appropriate queuing discipline we can claim some control over transmission delay. We can definitely control the propagation delay with our DTPF algorithm. And we can control queuing delays by classifying the traffic and applying TAPRIO queuing discipline. According to our results, the average delay in transmitting TSN packets from talker to listener is 7.45 μs. Fig.11 depicts the delay in TSN packet transmission.

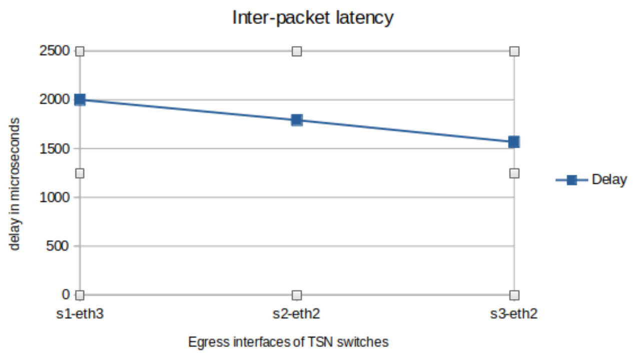


FIGURE 12. **Inter-packet latency at the egress port of TSN switches.**

1) INTER-PACKET LATENCY

The Fig.12 shows the behavior of inter-packet latency on the TSN nodes participating in TSN path/flow. We can observe slightly higher inter-packet latency on the interface s1-eth3 which is the egressing interface of the first TSN switch of our TSN path as depicted in Fig.12, as compared to the rest of the TSN switches. This is because packet classification is being done in userspace using the tc command at ingress interfaces. The internal implementation of the tc command uses netlink hooks to send specific messages to the kernel to update the configuration accordingly [40]. On the rest of the interfaces, the inter-packet latency is found to be stable.

VI. CONCLUSION AND FUTURE SCOPE

In this paper, the TSN architecture is realized using open-source tools with an SDN framework. The Linux-based queuing discipline is configured with multiple virtual queues. The results show that the egress ports of each TSN switch function as per the gating logic defined in Table 4. The delay in TSN data delivery and the inter-packet latency have been analyzed with results. Hence the overall observation on virtual queue performance are also promising which paves the way to use our unique architecture for PoC. The novel DTPF algorithm efficiently identifies the switches participating in a TSN flow. As a future work, this model can be further enhanced to generate an adaptive approach for rescheduling the TSN flows by selecting the optimal path among the multiple available backup paths.

REFERENCES

[1] C. Basumatary and H. S. Satheesh, "Queuing policies for enhanced latency in time sensitive networking," in *Proc. IEEE Int. Conf. Electron., Comput. Commun. Technol. (CONECCT)*, Jul. 2020, pp. 1–7.

[2] T. Ylonen, "SSH—Secure login connections over the internet," in *Proc. 6th USENIX Secur. Symp.*, vol. 37, 1996, pp. 40–52.

[3] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*, IEEE Standard 802.1Qcc-2018 (Amendment to IEEE Standard 802.1Q-2018 as amended by IEEE Standard 802.1Qcp-2018), Oct. 2018, pp. 1–208, doi: 10.1109/IEEESTD.2018.8514112.

[4] G. N. Kumar, K. Katsalis, P. Papadimitriou, P. Pop, and G. Carle, "Failure handling for time-sensitive networks using SDN and source routing," in *Proc. IEEE 7th Int. Conf. Netw. Softw. (NetSoft)*, Jun. 2021, pp. 226–234.

[5] S. B. H. Said, Q. H. Truong, and M. Boc, "SDN-based configuration solution for IEEE 802.1 time sensitive networking (TSN)," *ACM SIGBED Rev.*, vol. 16, no. 1, pp. 27–32, Feb. 2019.

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

[7] T. Kobzan, I. Blöcher, M. Hendel, S. Althoff, A. Gerhard, S. Schriegel, and J. Jasperneite, "Configuration solution for TSN-based industrial networks utilizing SDN and OPC UA," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2020, pp. 1629–1636.

[8] L. Lo Bello and W. Steiner, "A perspective on IEEE time-sensitive networking for industrial communication and automation systems," *Proc. IEEE*, vol. 107, no. 6, pp. 1094–1120, Jun. 2019.

[9] W. Quan, W. Fu, J. Yan, and Z. Sun, "OpenTSN: An open-source project for time-sensitive networking system development," *CCF Trans. Netw.*, vol. 3, no. 1, pp. 51–65, Sep. 2020.

[10] S. Schriegel, T. Kobzan, and J. Jasperneite, "Investigation on a distributed SDN control plane architecture for heterogeneous time sensitive networks," in *Proc. 14th IEEE Int. Workshop Factory Commun. Syst. (WFCS)*, Jun. 2018, pp. 1–10.

[11] T. Gerhard, T. Kobzan, I. Blöcher, and M. Hendel, "Software-defined flow reservation: Configuring IEEE 802.1Q time-sensitive networks by the use of software-defined networking," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2019, pp. 216–223.

[12] M. A. Metaal, R. Guillaume, R. Steinmetz, and A. Rizk, "Integrated industrial Ethernet networks: Time-sensitive networking over SDN infrastructure for mixed applications," in *Proc. IFIP Netw. Conf. (Networking)*, Jun. 2020, pp. 803–808.

[13] Y. Wang, J. Chen, W. Ning, H. Yu, S. Lin, Z. Wang, G. Pang, and C. Chen, "A time-sensitive network scheduling algorithm based on improved ant colony optimization," *Alexandria Eng. J.*, vol. 60, no. 1, pp. 107–114, Feb. 2021.

[14] L. Leonardi, L. L. Bello, and G. Patti, "Bandwidth partitioning for time-sensitive networking flows in automotive communications," *IEEE Commun. Lett.*, vol. 25, no. 10, pp. 3258–3261, Oct. 2021.

[15] G. N. Kumar, K. Katsalis, and P. Papadimitriou, "Coupling source routing with time-sensitive networking," in *Proc. IFIP Netw. Conf. (Networking)*, Jun. 2020, pp. 797–802.

[16] Y. Maleh, Y. Qasmaoui, K. El Gholami, Y. Sadqi, and S. Mounir, "A comprehensive survey on SDN security: Threats, mitigations, and future directions," *J. Reliable Intell. Environ.*, vol. 9, no. 2, pp. 201–239, Jun. 2023.

[17] W. Kong, M. Nabi, and K. Goossens, "Run-time recovery and failure analysis of time-triggered traffic in time sensitive networks," *IEEE Access*, vol. 9, pp. 91710–91722, 2021.

[18] M. Böhm, J. Ohms, and D. Wermser, "Multi-domain time-sensitive networks—An east-westbound protocol for dynamic TSN-stream configuration across domains," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2019, pp. 1363–1366.

[19] V. Balasubramanian, M. Aloqaily, and M. Reisslein, "An SDN architecture for time sensitive industrial IoT," *Comput. Netw.*, vol. 186, Feb. 2021, Art. no. 107739.

[20] M. Boehm, J. Ohms, M. Kumar, O. Gebauer, and D. Wermser, "Time-sensitive software-defined networking: A unified control-plane for TSN and SDN," in *Proc. Mobile Commun., Technol. Appl., 24th ITG-Symp.* Berlin, Germany: VDE, May 2019, pp. 1–6.

[21] S. Nam, H. Kim, and S. Min, "Simplified stream reservation protocol over software-defined networks for in-vehicle time-sensitive networking," *IEEE Access*, vol. 9, pp. 84700–84711, 2021.

[22] F. Prinz, M. Schoeffler, A. Lechler, and A. Verl, "Dynamic real-time orchestration of I4.0 components based on time-sensitive networking," *Proc. CIRP*, vol. 72, pp. 910–915, Jan. 2018.

[23] F. Rezabek, M. Bosk, T. Paul, K. Holzinger, S. Gallenmüller, A. Gonzalez, A. Kane, F. Fons, Z. Haigang, G. Carle, and J. Ott, "EnGINE: Flexible research infrastructure for reliable and scalable time sensitive networks," *J. Netw. Syst. Manage.*, vol. 30, no. 4, p. 74, Oct. 2022.

[24] N. Sambo, S. Fichera, A. Sgambelluri, G. Fioccola, P. Castoldi, and K. Katsalis, "Enabling delegation of control plane functionalities for time sensitive networks," *IEEE Access*, vol. 9, pp. 136151–136163, 2021.

[25] *Avnu's Use of 802.1 TSN Mechanisms for Industrial and Automotive Markets*, Standard IEEE 802.1 Plenary, Jul. 2016.

[26] J. Woods and S. Zuponcic, "QoS—Application of TSN to Ethernet/IP networks," in *Proc. ODVA Ind. Conf.*, 2017, pp. 1–16.

[27] *IEEE Standard for Layer 2 Transport Protocol for Time Sensitive Applications in a Bridged Local Area Network*, IEEE Standard 1722-2011, May 2011, pp. 1–65, doi: 10.1109/IEEESTD.2011.5764875.

[28] K. Weber. *EtherCAT Technology Group, EtherCAT and TSN—Best Practices for Industrial Ethernet System Architectures*. Accessed: Apr. 4, 2023. [Online]. Available: https://www.ethercat.org

[29] *Open vSwitch Quality of Service FAQ*. Accessed: May 24, 2023. [Online]. Available: https://docs.openvswitch.org/en/latest/faq/qos/

[30] *Configuring TC on Linux Control*. Accessed: Apr. 4, 2023. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html

[31] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, IEEE Standard 802.1Qbv-2015 (Amendment to IEEE Standard 802.1Q-2014 as amended by IEEE Standard 802.1Qca-2015, IEEE Standard 802.1Qcd-2015, and IEEE Standard 802.1Q-2014/Cor 1-2015), Mar. 2016, pp. 1–57, doi: 10.1109/IEEESTD.2016.8613095.

[32] D. Borkmann. *Net, Sched: Add Clsact*. Accessed: Jun. 2023. [Online]. Available: https://lwn.net/Articles/671458/

[33] P. Karachatzis, J. Ruh, and S. S. Craciunas, "An evaluation of time-triggered scheduling in the Linux kernel," in *Proc. 31st Int. Conf. Real-Time Netw. Syst.*, Jun. 2023, pp. 119–131.

[34] M. Kumar, "Evaluation of the time-aware priority queueing discipline with regard to time-sensitive networking in particular IEEE 802.1 Qbv," in *Proc. Int. Conf. Appl. Innov. IT*, vol. 7, no. 1. Köthen, Germany: Anhalt Univ. of Applied Sciences, 2019, pp. 1–6.

[35] N. S. Bülbül, D. Ergenç, and M. Fischer, "SDN-based self-configuration for time-sensitive IoT networks," in *Proc. IEEE 46th Conf. Local Comput. Netw. (LCN)*, Oct. 2021, pp. 73–80.

[36] B. Rother, M. Kasparick, E. Schweißguth, F. Golatowski, and D. Timmermann, "Automatic configuration of a TSN network for SDC-based medical device networks," in *Proc. 16th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, Apr. 2020, pp. 1–8.

[37] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of IEEE 802.1 TSN networks," in *Proc. 22nd IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2017, pp. 1–8.

[38] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "A configuration agent based on the time-triggered paradigm for real-time networks," in *Proc. IEEE World Conf. Factory Commun. Syst. (WFCS)*, May 2015, pp. 1–4.

[39] M. Bohm, J. Ohms, M. Kumar, O. Gebauer, and D. Wermser, "Dynamic real-time stream reservation for IEEE 802.1 time-sensitive networks with OpenFlow," in *Proc. 8th Int. Conf. Appl. Innov. IT* 2020, pp. 7–12.

[40] J. H. Salim, "Linux traffic control classifier-action subsystem architecture," in *Proc. Netdev*, 2015, pp. 1–10.

[41] M. Maliosz, I. Moldován, M. Máté, C. Simon, and J. Harmatos, "Deterministic local cloud for industrial applications," in *Proc. IEEE 19th Int. Conf. Factory Commun. Syst. (WFCS)*, Pavia, Italy, Apr. 2023, pp. 1–8, doi: 10.1109/WFCS57264.2023.10144237.

[42] M. Jayakumar. *Data Plane Development Kit*. Accessed: Jun. 20, 2023. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/dpdk-cookbook-759202.pdf

[43] Y. Han, S. Peng, and Y. Gao. (Jun. 11, 2023). *Analysis and Evaluation for TSN Queuing Mechanisms*. [Online]. Available: https://www.ietf.org/id/draft-hp-detnet-tsn-queuing-mechanisms-evaluation-00.html

[44] Y. Gou, X. Jiang, Z. Liu, R. Zhou, Q. Zhou, and G. Xie, "Optimization on architecture of time-sensitive software-defined network," Lanzhou Univ., Lanzhou, China, Tech. Rep., Apr. 2023.

[45] J. Jiang, Y. Li, X. Zhang, M. Yu, C. D. Lee, and S. H. Hong, "Assessing the traffic scheduling method for time-sensitive networking (TSN) by practical implementation," *J. Ind. Inf. Integr.*, vol. 33, Jun. 2023, Art. no. 100464.

[46] S. Buzura, A. Peculea, B. Iancu, E. Cebuc, V. Dadarlat, and R. Kovacs, "A hybrid software and hardware SDN simulation testbed," *Sensors*, vol. 23, no. 1, p. 490, Jan. 2023.

[47] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (TSN)," *IEEE Access*, vol. 11, pp. 61192–61233, 2023, doi: 10.1109/ACCESS.2023.3286370.

[48] D. Pannell, "Choosing the right TSN tools to meet a bounded latency," in *Proc. IEEE SA Ethernet IP@ Automot. Technol. Day*, 2019.

ASHA G. HAGARGUND (Graduate Student Member, IEEE) received the Bachelor of Engineering degree in electronics and communication and the Master of Technology degree in digital communications from Visvesvaraya Technological University, Karnataka, India. She is currently pursuing the Ph.D. degree with the National Institute of Technology Karnataka, Surathkal.

She is also an Assistant Professor with the BMS Institute of Technology and Management, India. She has more than 12 years of teaching experience and two years of industry experience. She has worked on projects related to mobile communication with Samsung India Software Operations, Bengaluru, India, and worked on security technologies with Ingersoll Rand, Bengaluru. Her current research interests include time-sensitive networking and the Industrial IoT. She has executed various projects on networking using tools, such as Mininet, OMNET++, NS2, and Qualnet. Previously, she has worked on time sensitive network realization using various simulators. She has delivered talks on TSN in industry and academia. She has publications on wireless sensor networks, time-sensitive networks, image processing, and the Internet of Things.

Prof. Asha is a member of the Additive Manufacturing Society of India.

NEELAWAR SHEKAR VITTAL SHET received the Bachelor of Engineering degree from MIT, Manipal, India, the Master of Engineering degree in communication system from IIT Roorkee, India, and the Ph.D. degree from the National Institute of Technology Karnataka (NITK), Karnataka, Surathkal, India.

He is currently a Professor and the Head of the Department with the Electronics and Communication Department, NITK. He has executed a project sponsored by the Centre for Wireless Networking Technologies (MHRD TAT) on building a model for tracking train position on tracks. He has more than 35 years of teaching experience with NITK. He has more than 30 research publications in the area of machine learning, wireless networks, and the Internet of Things. He has published book chapters on congestion control mechanisms in vehicular networks, envisioned network architecture for IoT applications, topology control in wireless sensor networks, and reliable cross-layer design for e-health applications.

Dr. Shet is a fellow of the Institution of Electronics and Telecommunication Engineers.

MURALIDHAR KULKARNI (Senior Member, IEEE) received the B.E. degree in electronics engineering from the University Visvesvaraya College of Engineering, Bengaluru, the M.Tech. degree in satellite communication and remote sensing from the Indian Institute of Technology Kharagpur, Kharagpur, and the Ph.D. degree in electronics and communication engineering from JMI Central University, New Delhi.

He was a Scientist with the Central Power Research Institute, Bengaluru, and an Aeronautical Engineer with Hindustan Aeronautics Ltd., Bengaluru. He was an Assistant Professor with the Department of Electronics and Communication Engineering, Visvesvaraya College of Engineering, and an Associate Professor with the Department of Electronics and Communication Engineering, Delhi Technological University, New Delhi. Then, he joined the Department of Electronics and Communication Engineering, National Institute of Technology Karnataka Surathkal, Mangaluru, as a Faculty Member, in 2008, where he was a Professor in electronics and communication engineering. He was the HoD of the ECE Department, the Chairperson of the Career Development Center, a member of the Board of Governors, and the Coordinator of the Center of Excellence (CoE) in Wireless Sensor Networks, National Institute of Technology Karnataka (NITK), Surathkal. He has published more than 80 research papers in international/national journals and conferences and also has to his credit several funded research projects and delivered keynote addresses/technical talks in international conferences and workshops. He has also authored/coauthored five very popular textbooks in the areas of microwaves and radars, DSP, information theory and coding, and analog digital communications. His current research interests include digital communications and networks (OFDM, WSN, SDN, congestion control, 5G networks, and the IoT), optical communications and networks (FSO), microwave, and antenna systems.

Dr. Kulkarni is a fellow of IETE and a Life Member of ISTE and CSI.