Decentralized Dynamic Gate Scheduling of IEEE 802.1Qbv Time Aware Shaper and a TSN Simulator for Tactile Cyber-Physical Systems

Kurian Polachan*, Chandramani Singh‡, and Prabhakar T V†
Indian Institute of Science, Bangalore, India
{*kurian, †tvprabs, ‡chandra,}@iisc.ac.in

***Abstract*—Cybersickness and control-loop instabilities are two main concerns in Tactile Cyber-Physical Systems (TCPS). TCPS applications demand stringent bounds on end-to-end latencies to avoid their occurrences. Traditional best-effort networks cannot guarantee packet latencies in the presence of external traffic. On the other hand, emerging deterministic networks such as IEEE 802.1 Time Sensitive Networks (TSN) can isolate time-critical flows from external traffic using IEEE 802.1Qbv Time-Aware-Shaper (TAS) to guarantee bounded end-to-end packet latencies. In this work, we devise a decenteralized dynamic scheduling protocol to configure non-overlapping gate slots in TAS enabled TSN switches to support TCPS flows. Our protocol supports plug-and-play operation of TCPS terminals with guaranteed minimal end-to-end packet latencies. Further, we also present PYTSN, an open-source discrete-event network simulator containing models of TAS enabled TSN switches and TCPS terminals. We use PYTSN to evaluate our proposed scheduling protocol. In particular, we show the isolation of TCPS flows from external traffic and plug and play operation of TCPS terminals.**

I. Introduction

A teleoperation system consists of a human operator controlling a remote slave device such as a robot over a communication network using different types of feedbacks such as audio, video, haptics, or a combination of these (See Figure 1).

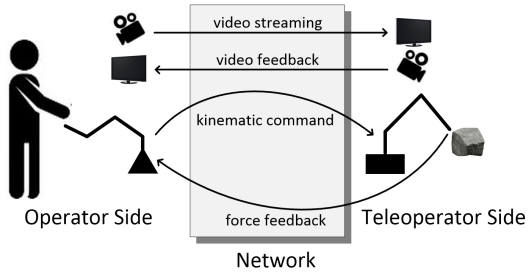

Fig. 1: A teleoperation system

Teleoperation systems have been around for several decades, but they still suffer from one main problem. Due to high latencies, jitter and packet drops in communication networks, existing teleoperation systems fail to support high operator hand speeds over large distances [1], [2]. At higher hand speeds, either the system's control loop may collapse [3], or the operator may experience cybersickness [4]. For critical applications, like telesurgery, the control loop collapse can cause the remote-side robot to go out of sync with the surgeon's hand, resulting in injuries to patients. Cybersickness, which occurs when feedback delays are notable, can result in physical and physiological effects in human operators, preventing them from using the teleoperation system for a long duration [5]. A possible solution to support high operator hand speeds in teleoperation systems is to build these systems with ultra-reliable low latency (URLL) communication networks, referred to as tactile internet, characterized by very low latency, jitter and packet drops [2], [6]. We refer to such teleoperation systems as Tactile Cyber-Physical systems (TCPS).

For a TCPS, the required transmits intervals and latencies for the kinematic, haptic and video packets depend on the end application. A transmit interval and end-to-end latency of $10\,\mathrm{ms}$ is considered adequate for many TCPS applications [7]–[9]. Here, network latencies contribute to only one part of the end-to-end latencies; end-to-end latency should accommodate various other delays such as delay in sensing, actuation, and data processing. Thus, the communication network delays are expected to be a small fraction of the end-to-end delay requirement of the TCPS flows, which could be around 10% of the end-to-end latency requirement. Consequently, in a TCPS, the kinematic, haptic and video packets may be transmitted at $10\,\mathrm{ms}$ but will require the network to guarantee a few hundred-microsecond packet latencies.

Tactile internet may be realized in several different ways. We explore the possibility of using IEEE 802.1 Time-Sensitive Networking (TSN) [10], [11]. TSN operates on Ethernet and uses several time-sensitive traffic shapers and schedulers to guarantee deterministic end-to-end latencies, negligible jitter and negligible packet loss for time critical flows. Among all the TSN traffic shapers, IEEE 802.1Qbv Time-Aware-Shaper (TAS) offers the best in class end-to-end latency to time-critical flows [12]. Our work focuses on scheduling time-critical TCPS flows in networks comprising of switches that support TAS.

A. Contributions and Outline

Our contributions are as follows.

- We develop a decentralized dynamic scheduling protocol to schedule TCPS flows in switches supporting TAS. The resultant schedules guarantee low end-to-end latencies for TCPS flows.
- We present PYTSN, an open-source discrete-event network simulator with simulation models for TSN-Switches

978-3-903176-32-4

and TCPS terminals to evaluate our decentralized dynamic scheduling protocol.

Outline of the Paper: This paper is structured as follows: Section II describes IEEE 802.1Qbv Time-Aware Shaper. Section III lists related work in the area of scheduling flows in time-sensitive networks and their limitation in supporting TCPS flows. Section IV describes our decentralized dynamic gate scheduling protocol. Section V presents PYTSN, our discrete-event TSN-Simulator . We evaluate PYTSN and the scheduling protocol in Section VI and conclude the paper in Section VII.

II. Background: IEEE 802.1Qbv Time-Aware Shaper

The principle behind the TAS is as follows. The output port of a TAS supported TSN-Switch has multiple queues. Figure 2 shows the case with three queues (q_0, q_1 and q_2). Each of these queues is time-gated. A queue is eligible to transmit its packets only when its gate is open. Each port has a Gate Control List (GCL) that tabulates the pattern of opening and closing of these gates. More precisely, the entries in the GCL decide which queue gates to open (q) and close ($\overline{q}$) and for how long (T_i), also called gate slot. For example, the first entry in the GCL shown in Figure 2, when applied, opens the gate of q_0 and closes the gate of the other two queues for T_0 seconds. The entries in the GCL are applied one after another from the start in a cyclic fashion. The repetition period is called the TAS Cycle Time (CT). In this arrangement, to isolate a time-critical flow, also known as Scheduled Traffic (ST), from other flows, an output queue is dedicated to the ST flow. The gate of this queue is open only after closing the gates of all other queues. If all the switches in the ST flow path follow this arrangement, and if the open/close time of the queue corresponding to the ST flow in all the switches in the path is time-synchronized, it becomes impossible for any other flows to affect the packet latencies of the ST flow. Further, such a configuration results in deterministic end-to-end packet latencies for the ST flows.

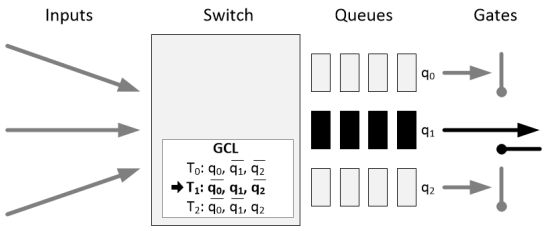

Fig. 2: Output port queues of a TAS supported TSN-switch. Figure shows a situation where the second GCL entry is applied, i.e., only the gate for q_1 is open.

III. Related Work

1) Scheduling Algorithms: Scheduling the gates of queues in TAS supported TSN-Switches to guarantee deterministic end-to-end latencies for ST traffic is an active research area [13]. Scheduling can be either static or dynamic. In static scheduling, a software tool determines the gate open and close times of the queues for different switches in a TSN network [14]–[16]. The network operator feeds the tool the details of network topology and the ST flow requirements. The tool then finds an optimal gate schedule for the operator to feed into a central TSN controller to configure the TSN switches. This approach is manual, does not support real-time addition and removal of ST flows and requires a centralized controller to manage TSN switches. Dynamic scheduling enables real-time addition and removal of ST flows [17], [18]. Dynamic scheduling is of two types, centralized and decentralized [19]. In a centralized method, a central controller orchestrates the real-time addition and removal of flows. Here, the central controller's presence makes this method costly and overly complex. On the contrary, in a decentralized method, the switches themselves take responsibility for scheduling flows without requiring a central controller, making it the most suitable method for small-scale dynamic networks.

We find only one work in the literature on a decentralized dynamic gate scheduling algorithm [17]. We find that this work's scheduling algorithm is tailor-made for ST flows, such as industrial process control flows that require very low transmitting intervals, low bandwidth requirements and fewer number of hops. We note that when we use the existing method for scheduling TCPS flows that require relatively high transmission intervals of $10\,\mathrm{ms}$ and $1\text{-}100\,\mathrm{Mbps}$ bandwidth [7], including video traffic, they result in higher-end-to end latencies. The following paragraphs explains this problem. *Throughout this paper, we refer to the existing method as the ST/BE Ratio Method.*

In the ST/BE Ratio Method, two gate slots exist in a CT, one for ST and other for Best Effort (BE) traffic. The gate slot allotted to ST is decided based on the number of registered ST flows passing through that switch. For example, if only one ST flow is registered, the ST gate slot of that switch is set to one-hop packet transmit time, calculated based on the ST packet size and packet transmit rate. If the transmitting terminal is synchronized with the first-hop switch, the delay the packets experience traversing through the first-hop switch will always be equal to one-hop packet transmit time. However, when the packet reaches the next-hop switch's input port, it may be blocked due to the second switch gate being closed for the ST flow and opened for a different class, e.g., BE traffic. The packet now will have to wait for one CT, which increases its latency. This latency increases with the number of hops; resulting in an end-to-end latency that is several multiples of CT. That said, such an approach is not problematic for industrial process control flows that have low transmit intervals, low bandwidth and fewer number of hops. A small CT such as $50\,\mathrm{us}$ will be sufficient to restrict the end-to-end packet latencies. However, for a TCPS, since video bandwidth requirement can be as high as $100\,\mathrm{Mbs}$ [7], a TAS gate slot of $\approx\ 1\,\mathrm{ms}$ is necessary to support one TCPS flow over a $1\,\mathrm{Gbps}$ network link. Which is possible if CT is $\approx\ 1\,\mathrm{ms}$ or more. However, CT cannot be more than $10\,\mathrm{ms}$ since TCPS transmits packets at $10\,\mathrm{ms}$ intervals. A CT of $1\,\mathrm{ms}$ can support one TCPS flow for a video bandwidth requirement of $100\,\mathrm{Mbps}$; to maximize the number of TCPS flows, we should set CT to its maximum permissible value of $10\,\mathrm{ms}$. But at this CT, the ST/BE Ratio method will result in packet latencies several folds higher than $10\,\mathrm{ms}$, not acceptable for TCPS flows.

2) Simulation Frameworks: Only a few simulators such as NeSTiNg, TSimNet, and CoRE4INET are designed to simulate TSN networks [20]–[23]. These simulators are either closed source or focus on simulating specific TSN standards, such as the IEEE 802.1Qbv TAS. However, simulating the TSN standards, such as IEEE 802.1Qbv, is only one part of evaluating dynamic gate scheduling algorithms. As described in Section V, dynamic gate scheduling algorithms' evaluation requires many changes in TSN-Switch's architecture and not just the transmission section alone. Existing papers on dynamic scheduling algorithms have not opened up the framework used for simulations; it is unclear if they have reused any existing TSN simulators or have created one from scratch. Since none of the TSN simulators in literature have TSN switches with support for dynamic gate scheduling algorithm, we decide to create one from scratch using Simpy. Simpy is an easy to use discrete event simulator based on Python [24]. [25] shows its use in simulating basic network components such as traffic generators, sources and queues. In our work, we use Simpy to create simulation models for several TSN network components such as the TAS supported TSN-Switch, with support for dynamic gate scheduling algorithm, TCPS and BE terminals, and network links.

IV. Decentralized Dynamic Scheduling Protocol

Our proposed decentralized dynamic scheduling protocol consists of two parts; a decentralized method to register/deregister TCPS flows and transfer data and a gate slot design method for allocating gate slots for TCPS flows in a CT. In this section, we briefly describe these methods.

A. TCPS Flow Registration and Data Transfer

Our proposed TCP flow registration method consists of two phases — an initialization phase and a run-time phase. The initialization phase starts with powering up of the network. Every switch runs a time synchronization protocol, e.g., IEEE 802.1AS, that synchronizes the time across the switches in the TCPS-TSN network [26], [27]. Once time synchronization is accomplished, every switch runs a link layer discovery protocol to learn the TCPS-TSN network topology and forwarding tables. We assume that every switch is equipped with necessary network protocols to complete the initialization phase. Once the network completes the initialization phase, the run-time phase begins. The run-time phase consists of a sequence of actions, for the TCPS terminals to register or deregister their flows with the network and transmit data as shown in Figure 3. We list these actions below. They repeat in a loop until the network is down.

1) Flow Registration: At the start of every CT, the switches send configuration beacon frames to the TCPS terminals that are directly connected to them but are yet to register their flows with the network. The configuration beacons are indicators that the switch is ready to accept connection requests from the attached TCPS terminals. The TCPS terminals that want to register their flow, send registration requests to their first-hop switches in response to the configuration beacons. A registration request consists of the TCPS flow requirements and the necessary information to identify the TCPS operator-teleoperator pair. After validating the request, the first-hop switch acts as a proxy for the TCPS terminal and broadcasts the registration request to all other switches in the network.

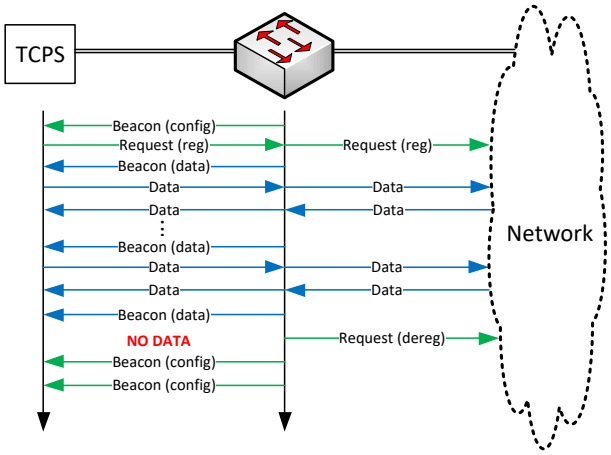

Fig. 3: TCPS flow registraiton and data transfer.

2) Processing of Registration Requests: After a predefined time from the start of a CT, all the switches in the network process the received registration requests. First, they identify the TCPS operator-teleoperator pairs and remove duplicate requests from the identified pairs. Next, they arrange the requests in the descending order based on the TCPS terminal IDs. The switches then allot gate slots to the TCPS pairs following the gate slot design strategy in Section IV-B. All the switches maintain an identical gate slot arrangement, which means that the switches allot gate slots to all the TCPS pairs, even though they may not be present in the flow path of certain TCPS pairs. Such a strategy, however, results in suboptimal bandwidth utilization in asymmetrical networks. We justify this strategy on the following grounds. First, it dramatically simplifies the gate slot allocation for TCPS pairs in a decentralized network. Second, any loss in bandwidth can be covered if the switches periodically check for traffic in the gate slots in run-time and, if found to be absent, admit BE traffic in such slots.

3) Acceptance/Denial of Flows: Sometimes the switches may have to deny registration requests from a few TCPS terminals, as there may not be enough room in the CT to accommodate all the flows. The switches must inform the TCPS terminals whether their flows are accepted.The switches communicate this through the beacons. If the registration request from a TCPS terminal is successful, then the terminal's first-hop switch stops sending configuration beacon frames to this terminal. Otherwise, the switch retransmits the configuration beacon in the next CT. The TCPS terminals after sending a registration request, check for presence of configuration beacon frames to determine whether their requests are accepted.

4) Data Transmission Phase: If a TCPS flow is successfully registered, the switch connected to the corresponding TCPS terminals sends data-beacon frames at the start of the TCPS flow's gate slots. The TCPS terminals send data packets in response to the data-beacon frames. The data-beacon frames help synchronize the TCPS terminal's data transfer with the gate slot allocated to them. They also eliminate the need for

the TCPS terminals to synchronize their clocks with the TSN switches.

5) Flow Deregistration: When a TCPS flow stops, the switches must be informed so that the allocated gate slots for the flow can be freed and could be allocated to other TCPS flows or to BE flows. We propose the following solution. The first-hop switch of a TCPS terminal can detect passively if the terminal sends data packets in response to the data beacon signals. If the switch does not detect data packets for K consecutive CTs, the switch can send a broadcast MAC frame at the start of the following CT requesting flow termination. The switches should process these deregistration requests along with the registration requests and should regenerate gate slot assignments before the next CT.

B. Gate Slot Design

In practice, it may be possible that TCPS pairs may have distinct requirements for transmit intervals, bandwidth and latency. Although our scheduling method could be extended to account for these requirements, in the current version of our work, for ease of explanation and demonstration, we consider a simplistic scenario. A scenario where all TCPS pairs in the network have identical transmit interval, bandwidth requirement and demand the smallest possible end-to-end latency. Our proposed gate slot design to support TCPS flows in such a scenario is as follows.

In every CT, the first gate slot is dedicated to network management traffic. Configuration beacon frames, registration and deregesitration requests are transmitted and received in this slot. Equation 1 decides, g_{nm}, the gate slot width for network management traffic. The gate slot width depends on the registration request frame sizes (s_{req}), network bandwidth (b_{nw})[1], the maximum number of TCPS pairs the network can support (n_{tcps}), the worst-case end-to-end delay for an ST packet between any two nodes in the TCPS-TSN network (d_{e2e}) and guard band equal to the transmit time of one BE packet.

$$g_{nm} = n_{tcps} \times s_{req}/b_{nw} + d_{e2e} + g_b \tag{1}$$

where,

$$n_{tcps} = \lfloor (b_{nw}/b_{tcps})(p/100) \rfloor$$

$$d_{e2e} = n_{hop}(d_{transmission} + d_{propogation} + d_{processing})$$

and b_{tcps} denotes the bandwidth requirement of the TCPS pairs. p is the percentage of the network bandwidth dedicated for TCPS flows. In our work, we use $p = 70\%$, this setting allots a minimum of 30% network bandwidth for the BE flows.d_{e2e} depends on the worst-case number of hops between any two nodes in the network, n_{hop}, transmit time of an ST packet, $d_{transmission}$, propagation delay of the ST packet over a link, $d_{propogation}$ and switch processing time, $d_{processing}$.

After allocating slot for network management traffic, the gate slot design block allocates gate slots for the TCPS pairs in the order in which the block receives the registration requests. The gate slot widths are set same for all pairs. We use Equation 2 for this. The setting ensures cut-through courses for all the TCPS flows in the network.

$$g_{tcps} = (b_{tcps}/b_{nw}) \times CT + d_{e2e} + g_b \tag{2}$$

When the gate slot design block receives a deregistration request, the associated gate slot is freed and assigned to BE traffic until a new TCPS registration request is received. The freed up slot is then reassigned to the new TCPS pair. Further, the gate design block ensures that at any given time, gate slots are allotted for only less than n_{tcps} no of TCPS pairs. Such a setting ensures that the percentage of gate slots dedicated to ST never exceeds 70% of the CT so that BE traffic gets a minimum of 30% of the CT.

Note: Registration/deregistration requests received in a CT are processed by the switches in the same CT to generate a new GCL; however, this GCL is activated only from the next CT onwards.

V. PYTSN: A TCPS-TSN Simulator

To evaluate our proposed dynamic scheduling protocol, we created PYTSN, a discrete event network simulator for the TCPS-TSN network. We created PYTSN's simulation framework in Python, and the simulation models using Simpy, a process-based discrete-event simulation framework based on Python [24]. PYTSN support simulation models for the most-common TCPS-TSN network components such as (i) TAS supported TSN-Switches, tsn_switch_ss and tsn_switch_ds with support for static and dynamic gate scheduling, respectively, (ii) TCPS and BE-Terminals, and (iii) network link. The simulation models expose several configure options for the user to tweak and simulate various network scenarios. This section describes the design of the TSN-Switch, *tsn_switch_ds*, that supports our dynamic scheduling algorithm along with the TCPS and BE terminals. We focus here only on the high-level architecture of these components. For the Simpy design of these components, refer to PYTSN's source code (see Section VII).

A. TSN-Switch

Figure 4 shows the architecture of *tsn_switch_ds*. The design supports four ports — Port-A, B, C and D. We can configure these ports as either type "ST" or "BE." Only ports configured as type "ST" can directly connect to a TCPS terminal and send beacons. Each port consists of an Rx and a Tx block. The Rx block receives incoming packets. It forwards the packet to the Port Control Logic (PCL) for further processing if it receives a packet containing a registration request. If it receives a data packet, it forwards the packet to the Switch Fabric (SF). SF routes the received packets from one port to another. To decide which port to route the packet, SF looks up the switch forwarding table using the incoming packet's destination-ID.

Figure 5 shows the architecture of the Tx block. It implements a dmux, several gated TAS queues and a priority transmitter. The current version of Tx has five queues dedicated to ST flows (q_1 to q_5), one for BE flow (q_{be}) and one network management queue (q_{nm}) to route registration packets. The dmux block uses the flow-id of the incoming packets to route

[1]Network bandwidth is the minimum of all link bandwidths in the LAN

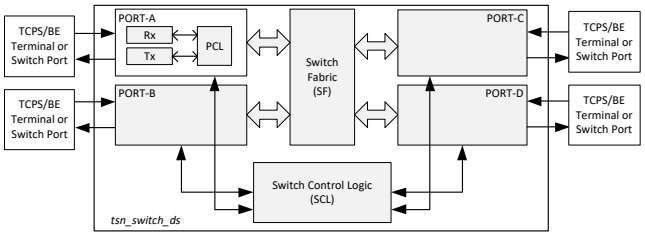

Fig. 4: Architecture of the TSN-Switch, *tsn_switch_ds*, which supports our decentralized dynamic gate scheduling algorithm. Each port of the switch can be connected to either a BE/TCPS terminal or to the port of a different switch using a full-duplex network link.

it to the associated queue. Every TCPS pair embeds a unique flow-id (> 0) in the packets it sends. All BE terminals embed a flow-id of -1 in its packets. Registration, deregistration and beacon packets uses a flow-id of 0 to distinguish it from packets from TCPS and BE terminals. Packets in a queue are transmitted when the gate of the queue is open. The block receives these gate control events from PCL. The priority transmitter uses the gate control to read packets from the open queues. It prioritizes their transmission using the packet's priority field if more than one queue is open at the same time. The Tx block is also used by PCL to route configuration and data beacon signals to the connected TCPS terminal.

Although the Tx-block appears to have simple architecture, building its discrete event simulation model is not trivial. First, we need to ensure that data is transferred from queues to the Simpy store resources (i_i) at the priority transmitter's input when there is no pending data in these store resource waiting for transmission. In our work, we use an interrupt mechanism; each time after transmitting data, the priority transmitter will generate an event noted by the Simpy process (*transfer-process*) that transfers data from the queue to the priority transmitter's input-side store resources. A similar problem exists when the priority transmitter tries to read data from its input-side Simpy store resources whose queue gates are open, but no packets are available. Here too, we implement an interrupt mechanism. Whenever packets are transferred to the Simpy store resources (i_i) or when there is a change in the gate control, an event is generated, which the packet transmitter waits to read packets.

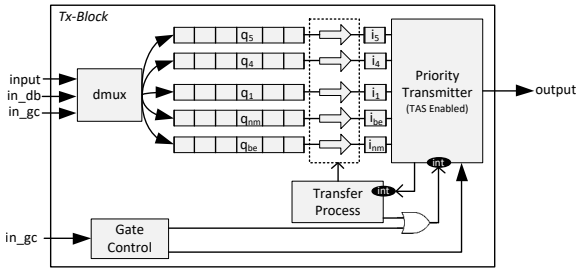

Fig. 5: Architecture of the Tx block in *tsn_switch_ds*

Note: The Tx block is fed data packets through its *input* port, the registration and deregistration packets through the *in_nm* port, beacon packets through the *in_db* port and gate control events from the *in_gc* port.

The PCL's residing in the switch ports (see Figure 4) generates gate control events to open and close gates of Tx block's queues. For generating the gate control events, it uses the GCL it receives from the Switch Control Logic (SCL) at the start of every CT. Further, PCL is responsible for (i) transferring registration requests received by the Rx block to SCL, (ii) transmitting the broadcast registration messages received from SCL to other switches in the TSN-Network via the Tx block, and (iii) generating and transmitting both configuration and data beacon to the connected TCPS terminal. PCL infers the timings to generate and send the beacons from the GCL it receives from SCL.

SCL (see Figure 4) controls and coordinates the functioning of our decentralized dynamic gate scheduling algorithm. It aggregates all the registration requests received from all the four ports and processes them to generate GCL. It sends the computed GCL to the switch ports' PCL blocks at the start of every CT. During the registration phase of TCPS terminals, SCL is also responsible for broadcasting the registration requests it receives to other connected switches in the TCPS-TSN network. For this, SCL forwards the registration packets it receives from any port to all other switch ports.

B. Terminals

1) TCPS Terminal: Figure 6 shows the design of the TCPS terminal. The purpose of this block is to simulate the registration and data transfer from/to TCPS terminals. TCPS terminal will send the registration packet in response to the configuration beacon it receives. Since the first-hop switches send configuration beacons in every CT, the terminal takes a maximum of one CT to register. After successful registration, it sends data packets in response to the data beacons it receives. The size and number of packets to send in response to each data beacon are configurable. Further, the terminal also logs received data packets using the traffic sink sub-block. Post simulation, the log is used for processing packet latencies and terminal reception rates. We also provide an option to enable/disable the output section of the block. We use this control in Section VI-D to demonstrate the dynamic plug and play operation of TCPS terminals.

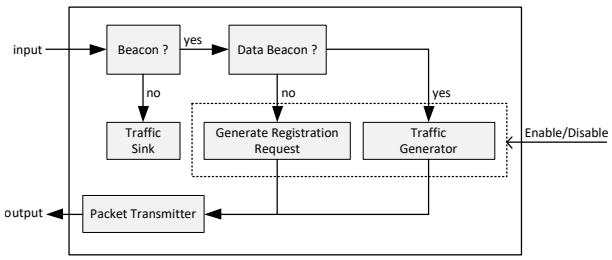

Fig. 6: Architecture of PYTSN's TCPS-Terminal.

2) BE-Terminal: A BE terminal design consists of two sub-blocks. A traffic generator blocks that generate and send best effort traffic and a traffic sink block that logs data packets it receives. Here too, we provide an option to enable/disable the

block's output section to simulate the dynamic plug and play operation of the terminal.

VI. Evaluation

In this section, as a first step, we validate PYTSN and its simulation models. We create a simple network topology in PYTSN using TAS supported TSN-Switches and conduct simulations for different QoS and traffic conditions. We then compare the simulation results with computed values to validate PYTSN's simulation models. Upon satisfaction that the simulator works as expected, we then use PYTSN to compare our proposed gate slot allocation method with the ST/BE ratio method in [17]. We show our method's superior performance at higher CT values in maintaining end-to-end packet latencies of ST flows under control. Finally, we demonstrate our decentralized dynamic scheduling algorithm. In particular, we demonstrate the isolation of TCPS flows from external traffic and plug and play operation of TCPS terminals.

Remark: Refer to PYTSN's GitHub page (see Section VII) to recreate the evaluation results in this section.

A. Demonstration and Validation of PYTSN

Figure 7 shows the evaluation setup. It consists of two TAS supported TSN-Switches, four BE terminals and two ST terminals interconnected using full-duplex links with propagation delay of $0.1\,\text{us}$. We assume that BE-Terminals 1 and 4, BE-Terminals 3 and 6 and ST-Terminals 2 and 5 communicate. The ST terminals exchange $100\,\text{B}$ packets every $1\,\text{ms}$. BE terminals exchange $100\,\text{B}$ packets at the rate of $7 \times 10^5\,\text{packets/s}$ with the interpacket times being i.i.d. exponential random variables. We fix packet transmission rates in the network to $1\,\text{Gbps}$, transmit-queue capacity of switches to $1000\,\text{packets}$ and switch processing time to $0.5\,\text{us}$. We simulate the network for the following QoS and traffic conditions.

- Conf-1: QoS=None, BE Traffic=Disabled
- Conf-2: QoS=None, BE Traffic=Enabled
- Conf-3: QoS=Priority Queuing, BE Traffic=Enabled
- Conf-4: QoS=TAS, BE Traffic=Enabled

For the conf-4, we fix the gate widths for ST and BE traffics to $50\,\text{us}$ and $950\,\text{us}$, respectively.

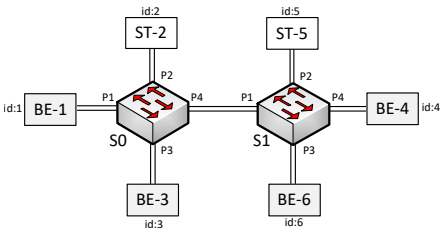

Fig. 7: Network Topology Created in PYTSN

Figure 8 show the latencies experienced by the packets traversing from ST-Terminal 2 to 5 for different configurations. Due to the network's symmetry, latencies experienced by packets from ST-Terminal 5 to 2 will have identical behaviour. In the absence of BE traffic or in the presence of TAS, i.e, conf-1 and conf-4, the end-to-end latencies of ST packets stays constant. We see that the observed latencies match the computed value of $3 \times d_{transmission} + 3 \times d_{propogation} + 2 \times d_{processing} = 3 \times 0.8\,\text{us} + 3 \times 0.1\,\text{us} + 2 \times 0.5\,\text{us} = 3.7\,\text{us}$[2]. With priority queuing in conf-3, the ST packet latencies vary between the $3.7\,\text{us}$ and $4.5\,\text{us}$. This range is also in line with the computed value. A BE packet already in transmission causes an ST packet to experience increased latency. The maximum increase in the latency corresponds to one BE packet's transmission time, which is $0.8\,\text{us}$ for a $100\,\text{B}$ packet. The maximum latency of an ST packet in conf-3 is thus $3.7\,\text{us} + 0.8\,\text{us} = 4.5\,\text{us}$. In the absence of TAS and priority queueing, i.e., in conf-2, the end-to-end latencies experienced by ST packets are severely affected by BE traffic since both ST and BE packets share the same transmit queue. When BE traffic is sufficiently high to throttle the network bandwidth, the ST packets' latencies shoot up until BE and ST packets (predominantly BE packets) start to drop due to the switches' limited queue capacity. Although the limited queue capacity of switches in conf-2 bounds the ST packet latencies, it is at the expense of packet drops, which is unacceptable for many applications.

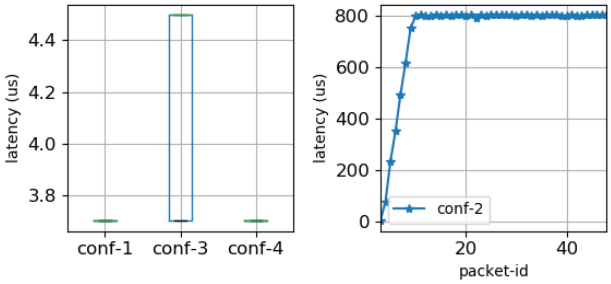

Fig. 8: Latencies experienced by ST packets traversing from terminal ST-2 to ST-5 for different switch configurations.

Note: Although priority queuing (conf-3) results in an increase in ST packet latency by only $0.8\,\text{us}$, we cannot rely on this method to guarantee bounded latencies in networks that carry many high priority ST flows.

B. ST/BE Ratio Method vs. Our Method of Gate Slot Design

In this experiment, we first assess the end-to-end latencies experienced by the ST flows when traversing through TAS supported TSN-Switches with gate slots designed as per the ST/BE ratio method described in [17]. We then compare these latencies with the latencies obtained using our method of gate slot allocation. We use the network topology in Figure 9 with ten TAS supported TSN-Switches for the PYTSN simulations. We fix the processing times of the switches to $0.5\,\text{us}$, packet transmission rates in the network to $1\,\text{Gbps}$, and the link propagation delay to $0.1\,\text{us}$. The gate slots of all these switches are time-synchronized. ST terminals 1 and 4, 2 and 5 and 3 and 6 communicate with each other with $100\,\text{B}$ packets every $10\,\text{ms}$.

TAS Gate Slot Design —

[2]Delay in transmitting ($d_{transmission}$) a $100\,\text{B}$ packet at a rate of $1\,\text{Gbps}$ is $0.8\,\text{us}$.

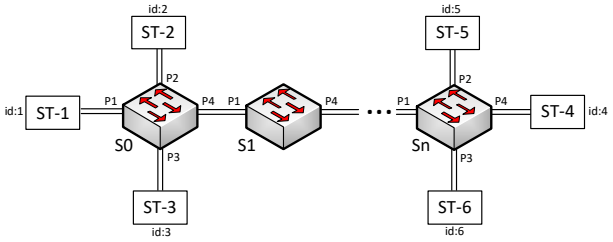

Fig. 9: Network topology created in PYTSN consisting of several TAS supported TSN-Switches. ST terminals 1 and 4, 2 and 5 and 3 and 6 communicate with each other 100 B of packet every 10 ms

ST/BE Ratio Method: In the ST/BE ratio method, only one ST slot exists in CT. The ST slot width in a switch is configured to the one-hop transmit time required to transmit all the packets corresponding to the switch's registered flows. Since, in Figure 9, each switch has to support three flows in each direction, the ST gate slots in the switches' egress ports should be wide enough to transmit three 100 B packets. We thus set the gate slot width to $3 \times 0.8\,\text{us} = 2.4\,\text{us}$, were 0.8 us is the time to transmit a 100 B packet at 1 Gbps. Further, each ST terminal's transmit time is synchronized with the first hop switch's ST slot. The gate slots in all the switches are also synchronized.

Our Method: In our method, each ST flow is assigned a separate ST slot in each CT, and transmission of each ST flow is synchronized to its ST slot. For the setup in Figure 9, we thus configure three ST slots in each CT. Each slot is wide enough to allow cut through traversal of ST packets between the transmitting and receiving ST terminals. For the example setup consisting of ten hops we fix each ST slot to $11 \times 0.1\,\text{us} + 11 \times 0.8\,\text{us} + 10 \times 0.5\,\text{us} \approx 15\,\text{us}$, where 0.1 us is the link propagation delay, 0.8 us is the time to transmit 100 B packet at 1 Gbps rate, and 0.5 us is the switch processing time.

We present the simulation result in Table I. We see that with the ST/BE ratio method, the end-to-end latencies of the ST packets increases with CT. At a CT of 10 ms, which is the necessary CT value for supporting TCPS flows, the latencies are significant and not acceptable to TCPS flows. On the other hand using our method, the end-to-end latencies stay at considerably lower levels of 14.9 us.

TABLE I: (Minimum, Maximum) end-to-end latency's experienced by ST packets from ST-1,2,3,4,5 and 6 for different gate slot allocation methods.

Method	Latency (us)
ST/BE Ratio (CT=50us)	(204, 252)
ST/BE Ratio (CT=1ms)	(4003.7,5002.3)
ST/BE Ratio (CT=10ms)	**(40003.7,50002.3)**
Our Method (CT=10ms)	**(14.9, 14.9)**

C. Performance Evaluation of PYTSN

In this experiment, we profile the memory usage and runtime of PYTSN. We use the network topology in Figure-9 with a variable number of TAS supported TSN-Switches for the simulations. We use the ST/BE ratio method for GCL slot allocation as it allows us to run simulations using low CTs and thus high terminal transmit rates. We enable all six ST flows (three in each direction). We run the simulation on a machine with two Intel Xeon 4116 processors at a clock speed of 2.1 GHz and a total RAM of 1 GB, running Linux with kernel version 5.4.0. We run experiments for two different ST transmit intervals, 50 us and 10 ms, and different numbers of switches in the linear topology. For each case, we run simulations for 100 ms. We use the command-line tool *memory-profiler* to determine the memory consumption of PYTSN simulations [28].

We present the results in Figure 10. In PYTSN, memory is consumed mainly by the switch modules. We see that memory usage increases as the number of switches in the topology increases. ST transmit intervals have little effect on memory usage. We see that memory consumption for a topology with fifty switches (# of flows x switches = $6 \times 50 = 300$) is under 100 MB, indicating that we can run PYTSN on most common computing platforms. Regarding simulation runtime, in discrete event simulations, simulation time increases with the number of event occurrences. Lower transmit intervals, and a large number of switches result in more events, increasing the simulation time. We observe the same pattern in PYTSN simulations. The simulation runtimes increase with the number of switches in the topology and are larger for lower ST transmit intervals.

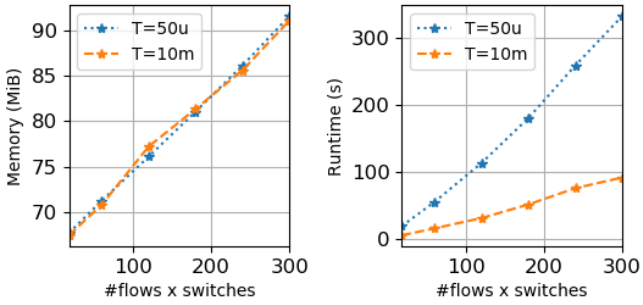

Fig. 10: Memory consumption and runtime of PYTSN simulations for different ST transmit intervals (T) and number of switches.

D. Evaluation of Our Decentralized Dynamic Gate Scheduling Algorithm

This experiment intends to demonstrate the functioning of our decentralized dynamic gate scheduling algorithm for different network configurations. We use the network topology in Figure 11 where all the end-nodes are associated with either TCPS or BE terminals. Table II lists these assignments, TCPS terminal pairs and BE sources and destination. We run the experiments for two network configurations. In the first configuration, the TCPS terminal pairs are enabled all together at the start of the simulation. In the second configuration, the TCPS terminal pairs are enabled one by one at intervals of 10 ms, starting from the simulation time of 10 ms. We use the

second configuration to simulate the dynamic plug and play operation of TCPS terminals. We demonstrate that the switches in the network can register the TCPS flows and allocate slots in both configurations. We show that end-to-end latencies of the TCPS terminal pairs are constant and are within the required specifications. We show that the BE terminals get the network bandwidth that is not used by TCPS traffic. We use the PYTSN switch module, *tsn_switch_ds*, that supports our dynamic scheduling algorithm for the simulations. We fix CT to $10\,\mathrm{ms}$, packet transmission rates in the network to $1\,\mathrm{Gbps}$, switch processing time to $0.5\,\mathrm{us}$, and the link propagation delay to $0.1\,\mathrm{us}$. We configure the TCPS terminals such that in each CT, the TCPS terminals send $1250\,\mathrm{packets}$ each of size $100\,\mathrm{B}$ to simulate a terminal transmit rate of $100\,\mathrm{Mbps}$. We configure the BE terminals to send data with interarrival times matching a Poisson process with a rate of $7 \times 10^5\,\mathrm{packets/s}$, which is sufficient to throttle the network bandwidth.

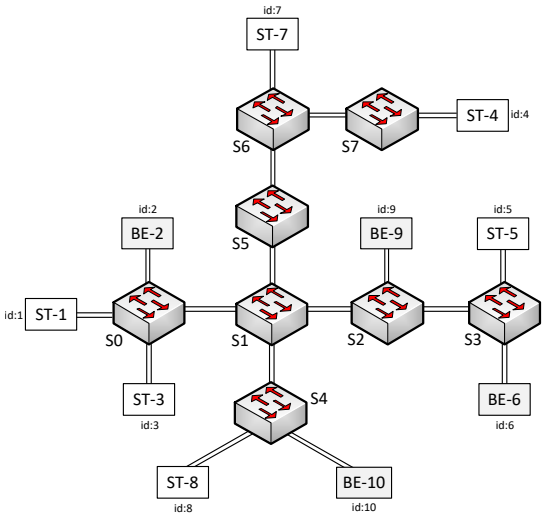

Fig. 11: Network topology created in PYTSN for evaluating our decentralized dynamic gate scheduling algorithm.

TABLE II: End node assignments of the network in Figure 11.

TCPS-Pairs	[3,7], [8,5], [1,4]
BE-Traffic	2, 6, 9 $\longrightarrow$ 10

Table III shows the end-to-end latency experienced by the TCPS and the BE packets. We see that for the TCPS packets, the end-to-end latencies remain constant and are only a few microseconds irrespective of the BE traffic. We see that the BE packet latencies are affected since all the three BE sources send high traffic to the destination BE-10. Figure 12 shows the rate at which BE-10 and TCPS terminals 3,7,8,5,1 and 4 receive packets. We see that the reception rate for the BE terminal is the highest in the first gate cycle because, in this cycle, there is only one gate slot assigned for the scheduled traffic — ST slot for the network management traffic for broadcasting registration requests. The rate decreases in the next cycle and stays the same for the first configuration. For the second configuration, the rate gradually decreases with the TCPS pairs getting plugged to the network. The rate reaches the steady-state once the network registers all the TCPS pairs.

TABLE III: End-to-end latencies (min, max, std.dev) experienced by TCPS and BE packets.

Conf-1		Conf-2	
src-dst	latency (us)	src-dst	latency (us)
3 $\leftrightarrow$ 7	(6.5, 6.5, 0)	3 $\leftrightarrow$ 7	(6.5, 6.5, 0)
8 $\leftrightarrow$ 5	(6.5, 6.5, 0)	8 $\leftrightarrow$ 5	(6.5, 6.5, 0)
1 $\leftrightarrow$ 4	(7.9, 7.9, 0)	1 $\leftrightarrow$ 4	(7.9, 7.9, 0)
2,6,9 $\rightarrow$ 10	(28.5, 4712.5, 1134)	2,6,9 $\rightarrow$ 10	(28.7, 4713.2, 952)

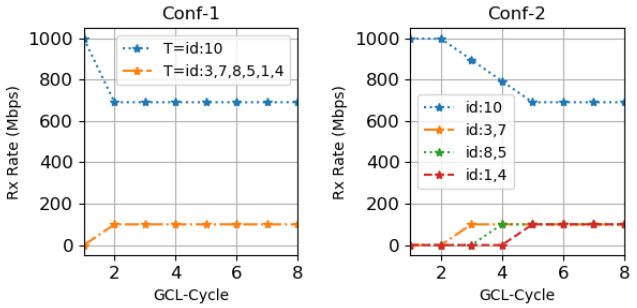

Fig. 12: Rates at which BE (id:10) and TCPS (id:3,7,8,5,1 and 4) terminals receives packet.

Note: Although we use three separate queues to support the three TCPS pairs, it is not mandatory. A single queue can support these TCPS pairs since the packet transmission times of these flows never overlap. However, separate queues help protect end-to-end latencies in the presence of misbehaving TCPS flows occupying more than the specified bandwidth.

VII. Conclusion

We find that existing decentralized dynamic gate scheduling protocols for IEEE 802.1Qbv Time-Aware-Shaper are suited only for applications such as industrial processes that have extremely low packet transmit intervals and bandwidth. They are not suitable for time-sensitive teleoperation systems such as Tactile Cyber-Physical Systems (TCPS) that demand relatively higher transmission intervals and high bandwidth. We propose a decentralized dynamic scheduling protocol to circumvent this problem. The simulation studies show that our method can provide end-to-latencies for TCPS flows that are many-fold times less compared to existing methods. Our method simplifies the design of TSN-Switches, requiring them to have a single Gate Control List (GCL). However, having a single GCL per switch constraints the switches' flexibility to schedule more than one flow in a given GCL slot. Solving this problem constitutes our future work. In contrast to existing network simulators build using C/C++, our simulator, PYTSN, is developed in Python. Since Python is a more user-friendly language than C/C++, we expect PYTSN to be easy for a beginner to use and extend. Further, we have also made the source code of PYTSN open through the GitHub page https://github.com/kpol-iisc/PYTSN.

REFERENCES

[1] D. Van Den Berg, R. Glans, D. De Koning, F. A. Kuipers, J. Lugtenburg, K. Polachan, P. T. Venkata, C. Singh, B. Turkovic, and B. Van Wijk, "Challenges in haptic communications over the tactile internet," *IEEE Access*, vol. 5, pp. 23 502–23 518, 2017.

[2] ITU. (2014) The tactile internet. [Online]. Available: https://www.itu.int/en/ITU-T/techwatch/Pages/tactile-internet.aspx

[3] K. Antonakoglou, X. Xu, E. Steinbach, T. Mahmoodi, and M. Dohler, "Toward haptic communications over the 5g tactile internet," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3034–3059, 2018.

[4] K. Polachan, T. V. Prabhakar, C. Singh, and D. Panchapakesan, "Quality of control assessment for tactile cyber-physical systems," in *2019 IEEE International Conference on Sensing, Communication and Networking (IEEE SECON)*, 2019.

[5] J. J. LaViola, Jr., "A Discussion of Cybersickness in Virtual Environments," *SIGCHI Bull.*, vol. 32, no. 1, pp. 47–56, Jan. 2000. [Online]. Available: http://doi.acm.org/10.1145/333329.333344

[6] A. Aijaz, Z. Dawy, N. Pappas, M. Simsek, S. Oteafy, and O. Holland, "Toward a Tactile Internet Reference Architecture: Vision and Progress of the IEEE P1918.1 Standard," 2018.

[7] S. K. Sharma, I. Woungang, A. Anpalagan, and S. Chatzinotas, "Toward tactile internet in beyond 5g era: Recent advances, current issues, and future directions," *IEEE Access*, vol. 8, pp. 56 948–56 991, 2020.

[8] J. Sachs, L. A. Andersson, J. Araújo, C. Curescu, J. Lundsjö, G. Rune, E. Steinbach, and G. Wikström, "Adaptive 5g low-latency communication for tactile internet services," *Proceedings of the IEEE*, vol. 107, no. 2, pp. 325–349, 2018.

[9] K. S. Kim, D. K. Kim, C. Chae, S. Choi, Y. Ko, J. Kim, Y. Lim, M. Yang, S. Kim, B. Lim, K. Lee, and K. L. Ryu, "Ultrareliable and low-latency communication techniques for tactile internet services," *Proceedings of the IEEE*, vol. 107, no. 2, pp. 376–393, 2019.

[10] 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.

[11] N. Finn, "Introduction to time-sensitive networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 22–28, 2018.

[12] I. S. Association *et al.*, "Ieee standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016.

[13] W. Steiner, S. S. Craciunas, and R. S. Oliver, "Traffic planning for time-sensitive communication," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 42–47, 2018.

[14] A. C. T. dos Santos, B. Schneider, and V. Nigam, "Tsnsched: Automated schedule generation for time sensitive networking," in *2019 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2019, pp. 69–77.

[15] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in ieee 802.1 qbv time sensitive networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 183–192.

[16] S. S. Craciunas, R. S. Oliver, and W. Steiner, "Demo abstract: Slate xns–an online management tool for deterministic tsn networks," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 103–104.

[17] A. Nasrallah, V. Balasubramanian, A. Thyagaturu, M. Reisslein, and H. ElBakoury, "Reconfiguration algorithms for high precision communications in time sensitive networks: Time-aware shaper configuration with ieee 802.1qcc (extended version)," 2019.

[18] A. Nasrallah, V. Balasubramanian, A. Thyagaturu, M. Reisslein, and H. ElBakoury, "Reconfiguration algorithms for high precision communications in time sensitive networks," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019, pp. 1–6.

[19] R. Hummen, S. Kehrer, and O. Kleineberg, "Tsn–time sensitive networking," *Hirschmann, USA, WP00027*, 2016.

[20] J. Jiang, Y. Li, S. H. Hong, A. Xu, and K. Wang, "A time-sensitive networking (tsn) simulation model based on omnet++," in *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE, 2018, pp. 643–648.

[21] P. Heise, F. Geyer, and R. Obermaisser, "Tsimnet: An industrial time sensitive networking simulation framework based on omnet++," in *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2016, pp. 1–5.

[22] M. Pahlevan and R. Obermaisser, "Evaluation of time-triggered traffic in time-sensitive networks using the opnet simulation framework," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 2018, pp. 283–287.

[23] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel, "Nesting: Simulating ieee time-sensitive networking (tsn) in omnet++," in *2019 International Conference on Networked Systems (NetSys)*, 2019, pp. 1–8.

[24] Simpy. (2020) Simpy-discrete event simulation for python. [Online]. Available: simpy.rtfd.org

[25] G. Networking. (2020) Basic network simulations and beyond in python. [Online]. Available: https://www.grotto-networking.com/DiscreteEventPython.html

[26] I. S. Association *et al.*, "Ieee standard for local and metropolitan area networks—timing and synchronization for time-sensitive applications in bridged local area networks," *IEEE Std*, vol. 802, 2020.

[27] K. B. Stanton, "Distributing deterministic, accurate time for tightly coordinated network and software applications: Ieee 802.1 as, the tsn profile of ptp," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 34–40, 2018.

[28] memory-profiler. [Online]. Available: https://pypi.org/project/memory-profiler/