Software-Defined Time Sensitive Networks (SD-TSN) for Industrial Automation

Mohamed Seliem
School of Computer Science and IT
University College Cork
Cork, Ireland
m.seliem@cs.ucc.ie

Dirk Pesch
School of Computer Science and IT
University College Cork
Cork, Ireland
d.pesch@cs.ucc.ie

***Abstract*—Industrial networks interconnect devices to support industrial processes and manufacturing systems. Industrial applications results in time-critical and/or mission-critical data traffic, which require high reliability, guarantees on the delay bounds and robust time synchronization with high precision. Time Sensitive Networking (TSN)is considered as a suitable multi-service network technology for industrial systems and applications. TSN provides greater timing accuracy, enhanced performance in reducing packet delay variation, and adds more networking determinism. However, the configurations of TSN switches presents a challenge (e.g. time-aware traffic shaping schedules). Each output port of a TSN switch has to follow timed schedule to achieve the desired performance. The schedules have to be distributed at the network initialization and have to be maintained upon new devices joining or obsolete device being removed. In this paper, we use the Software Defined Networking (SDN) paradigms to overcome the aforementioned challenges. We propose the Software-Defined Time Sensitive Networks (SD-TSN), where end devices sign contracts with the network controller, to check the availability of the network resources and to select an optimized path for the data transmission. Therefore, It achieves overall improved network resources management, while meeting QoS requirement with high reliability. Our performance analysis show that SD-TSN accurately schedule the critical time traffic resulting in a bounded end-to-end latency with very low jitter. Even in the presence of different types of data traffic, the critical data traffic meets the quality of service requirements with a maximum jitter of 6 μs. Moreover, SD-TSN takes 10 seconds run-time, compared to 2 minutes run-time when traditional TSN scheduling is used, to calculate the schedules of 100 TSN critical flows.**

***Index Terms*—Time-sensitive network, TSN, Software-defined networking, SDN, Industrial automation, mininet.**

I. Introduction

Industry 4.0 [1], the fourth industrial revolution, revolves around the use of Cyber-Physical Systems (CPS). A cyber-physical system consists of distributed digital devices, such as sensors and actuators, to measure, monitor and control a physical process. Those devices form a device layer, where they communicate with each other and with the devices that belong to the other CPSs. In addition, the data, collected by the sensing devices, has to be carried up to the applications layer; Same for the actions, to control the actuating units, have to be issued by the applications layer. Therefore, CPSs rely on communication and network layer to transport data and for control among them and other systems. Both applications layer and devices layer add limitaions, constraints and requirements on the communication and network layer to support different types of data traffic with different quality of services. For example industrial applications, It has strict timing and reliability requirements. Therefore, industrial networks has to be designed quite differently from conventional networks [2] [3]. Nondeterministic Network delay and jitter could significantly impact the control quality of a CPS. For example, isochronous motion control systems require extremely bounded jitter ($\approx$ 1 to 10 μs) for stability [4].

Traditionally, Real Time Ethernet (RTE) has been used in industrial networks with additional protocols to satisfy the specific needs of industrial application for determinism, reliability, and efficient communication with bounded delay and delay variance (jitter). RTE uses a modified Media Access Control (MAC) layer to achieve very low latency and deterministic responses. Protocols for Industrial Ethernet include EtherCAT [Beckhoff], EtherNet/IP [Rockwell], PROFINET [Siemens and GE], POWERLINK [B&R], and Sercos III. Using such protocols ensure the following:

- 1 Gbit/s data transmission rate over Ethernet.
- Select from widely available Cat5e/Cat6 cables
- Support optical fibres communication as a physical layer.
- Support wired and wireless standard networking hardware and protocols.
- Support multi-point links (multiple nodes are connected using the same link).

However, RTE [5] faces a principal challenge when migrating an existing system to implement a new protocol. Moreover, TCP based application protocols affects the real-time performance. Additionally, the minimum Ethernet frame size is 64 bytes, while typical industrial communication data sizes can be closer to 1–8 bytes. Finally, data transmission become less

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 16/RC/3918.

efficient because of the protocol overhead. IETF DetNets Working Group [6] and the IEEE 802.1 Time-Sensitive Networking (TSN) Task Group (TG) have worked on a set of standards to define mechanisms for the time-sensitive transmission of data over deterministic Ethernet networks, promising hard real-time capabilities. In addition, the standards address the aforementioned issues in real-time automation and industrial networks. IEEE TSN standards discuss three main functionalities:

1) **Time synchronization:** both end devices (hosts) and network devices (switches) need to have accordance of time to support real-time data transmissions. IEEE TSN-WG puts froward the IEEE 802.1AS , a tightly constrained subset of the IEEE 1588 Precision Time Protocol (PTP), to synchronize all the network nodes. IEEE 802.1AS uses master clock source to regulate the time synchronization frames through the network itself.
2) **Scheduling and traffic shaping:** to support different traffic classes [e.g. critical data traffic, audio, video, best effort… etc] with different priorities and different requirements. Traffic shaping is required to distribute frames/packets easier based on time or credit to smooth traffic. Therefore, it avoids buffers overflow along the network paths. The standards introduce many scheduling and shaping techniques such as IEEE 802.1Q strict priority, IEEE 802.1Qav credit-based traffic shaping, IEEE 802.1Qbv time-aware shaping, and IEEE 802.1Qcr asynchronous shaping.
3) **Selection of communication paths, path reservations and fault-tolerance:** traffic of high priority is replicated to be transmitted across multiple paths. In addition, duplicate removal is applied at or close to the packet destination. Moreover, resources reservation is considered to enhance the packet delivery and to achieve fault-tolerance.

Although the TSN standards provide solutions for real time data transmission over wired networks, several challenges remain, which we target in our work. For example, the time aware shaping configurations to guarantee a desired upper bound on network delay and to achieve the desired overall operation for different use cases. In SD-TSN we utilize software defined network paradigm to facilitate the network configurations and schedules distribution. SDN supports the deployment of network applications to execute central algorithms that provide forwarding decisions to control forwarding logic, configure resources to manage TSN queues and shaping parameters, and run resource utilization algorithms to monitor network performance. We run experiments over an emulated environment that is built using Mininet to demonstrate the operation of our design [7].

In detail, we make the following contributions:

- We introduce the problem of traffic scheduling in industrial applications and propose the use of network-device contracts to support computing and distributing traffic schedules to guarantee upper bounded latency for critical traffic.
- We deploy our SD-TSN implementation on the Mininet network emulation platform to assess the performance of end-devices in complying with the computed transmission schedules.
- We show through our evaluation that SD-TSN can meet real-time network requirements for realistic network sizes.

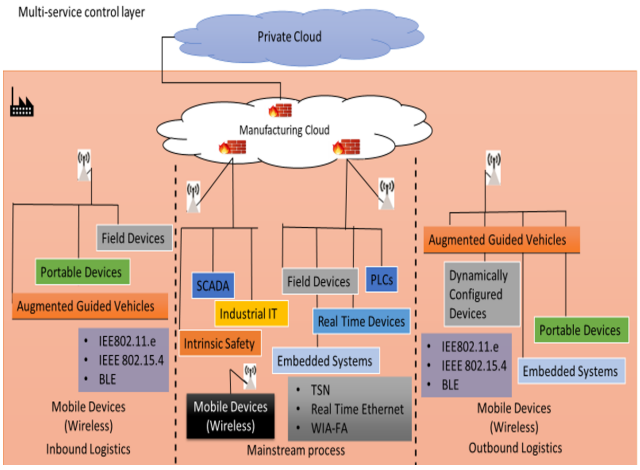

Fig. 1. Multi-Service Control Layer of a Factory Automation Model

The remainder of this paper is structured as follows. We present the related work in Section II. In Section III, we present the system model of SD-TSN and a Smart Factory use case. In Section IV, we present the process to compute and distribute the transmission schedules based on end devices and controller contracts. We present the environment setup on Mininet and evaluate our work in Section V. Finally, we present conclusions and future work in Section VI.

II. Related Work

Industrial automation [8] is built in top of a supporting network layer that interconnects complex systems to monitor, configure and control field devices. Such a complex network design creates an interesting set of challenges such as: producing timely correct decisions, synchronizing multi-core systems with different clock rates, transmitting deterministic data frames, reducing process time for queued frames in outgoing buffers, and sending frames no later than or at a certain time T. IEEE 802.1 TSN and IETF DetNets specified time-synchronized low latency streaming over layer 2 bridged networks, and deterministic data paths with bounds on packet latency, loss, and jitter over Layer 3 routed networks. However, the real time networks configuration is a complex problem as has been highlighted in [9]. Several studies discussed, via use cases, the benefits that SDN could bring to the configuration and management of industrial networks.

Authors in [10] provide a proof of concept for applying SDN management concepts to Real Time Ethernet. Their work

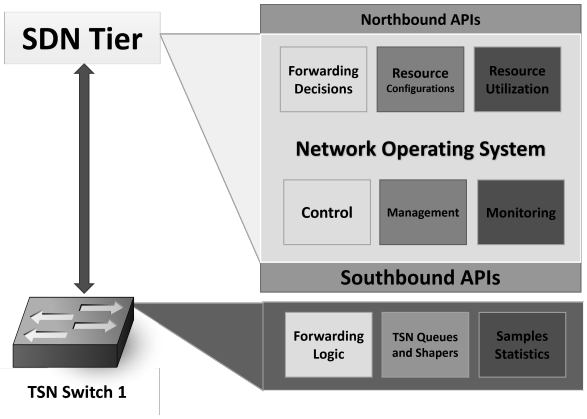

Fig. 2. Proposed SD-TSN Architecture

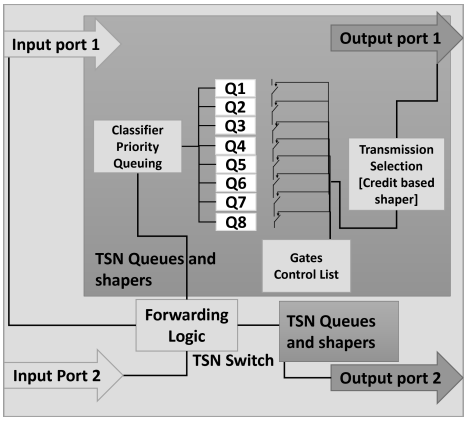

Fig. 3. TSN switch internal components

focused on integrating the Open Flow protocol and Power link real time Ethernet to overcome the issue of data loss due to link failure. Authors in [11] propose a complete SDN solution to make use of all TSN features not only to provide data path management but also to provide accurate time synchronization configuration. The aforementioned studies did not cover the complexity aspects of traffic scheduling which is considered an NP-hard problem (Bin-packing) [12]. The authors in [13] addressed this aspect through various Integer Linear Program (ILP) formulations to compute routes and transmission schedules for time-triggered flows. Moreover, they provide a proof of concept for their proposed TSSDN with virtually constant end-to-end latency (std. dev. $\leq$ 1µs) with worst case jitter $\leq$ 7µs. However, the authors have not considered time-triggered flows and their incremental behavior especially in industrial environments. Unlike the above studies, we are proposing SD-TSN, which not only allows to automatically configure time sensitive networks, but allows a TSN to interact with the network operating system through the use of contracts to reduce the complexity of the scheduling problem. Moreover, our approach enhances the network utilization factors by passing traffic flow related information that will affect the scheduling decisions. In addition, we are evaluating our SD-TSN proof concept in the Mininet network emulator for accurate evaluation, which we will make available to the community as an additional contribution.

III. Factory Automation Model And System Architecture

A typical industrial network environment integrates different nodes to perform different control and protection functions, transmit and receive different data flows, monitor and visualize information across the network. To model such an environment, Figure 1 depicts a multi-service control layer of a factory automation model where the main stream process comprises heterogenous technologies and equipment, such as SCADA, PLC, field and real time devices. Moreover, Figure 1 also contains automated guided vehicles (AGV) and wireless communication links, as well as other devices, such as personal computers, printers, cameras, etc. The factory control layer is connected to a manufacturing cloud to exchange data with external networks, avail of large data storage, add cloud processing power, and to provide higher security capabilities. As described in [3], industrial networks have deep architecture with many integrated devices, as even small industrial networks might have a three or four level hierarchy, possibly organized as a combination of different topologies (ring and star). The variety of these integrated devices, with different communication characteristics and a lack of interoperability, results in vendor-specific management systems. Moreover, the varying bandwidth or network load affects the data transfer performance. Therefore, resource monitoring is essential to maintain system performance, through considering the network state information as a driving factor of control functions to make dynamic decisions.

In the proposed architecture (Figure 2), we cover both Layer 3 and Layer 2 in the system model. Layer 3, the SDN tier, is based on a Network Operating System (NOS) [14], which mainly consists of three functionalities.

1) **Control plane:** composed of an OpenFlow [15] network controller that is responsible for setting up forwarding rules and a TSN application controller that allows each application end devices and TSN switches to make independent reservation decisions using the selected protocols. The control plane contains:
   - Discovery service: the network controller periodically uses the link layer discovery protocol to broadcast discovery packets to all TSN switches and waits for response packets, through which it can identify the network topology. End devices can be identified at first when making a contract to reserve a path for their traffic or later through examining their injected packet-in messages.
   - Path computation: the network controller determines

network routes to control the traffic and prevents loops.
- Dynamic control of traffic reservations: the TSN application controller orders end devices to sign a contract and make reservations, or it consults the network controller for reservations on behalf of the end devices while TSN switches defer reservation decisions to the network controller. (IEEE P802.1Qcc.)

2) **Resource management:** The Open vSwitch Database Management Protocol (OVSDB) carries out management and configuration operations on network devices as it allows the creation, configuration and deletion of ports, queues and tunnels of switches. The management protocol defines these network characteristics:
   - select topology control for both non-TSN and TSN traffic.
   - Maximum bandwidth allocation for TSN flows over each physical link [IEEE Std 802.1Q and/ or IEEE Std 802.1Qca].
   - Timing and synchronization protocols and parameters [IEEE Std 802.1AS-2011].
3) **Resource monitoring:** distributed sFlow agents, embedded in TSN switches, continuously monitor network traffic. They constantly report samples statistics, sFlow datagrams, to the sFlow collector. sFlow uses Packet-based sampling and Time-based sampling to create a network traffic visibility picture.

Layer 2 is based on Layer 2 TSN-switches. As shown in figure 3, TSN switches have forwarding logic, sFlow agent in addition to TSN priority queues and traffic shapers which basically consists of 4 parts:

1) Classifier: this entity uses a three-bit Priority Code Point (PCP) to decide which traffic class the packet belongs to [e.g. non-TSN traffic, TSN Traffic].
2) Priority Queues: each queue holds different types of traffic with different priority. Real time traffic [TSN] is assigned the highest priority, other types of traffic are assigned less priority.
3) packets transmission. If the gate of a queue is open, the first packet in the queue is eligible for transmission. If the gate is closed, the queue cannot transmit. GCL holds the entries for controlling each gate. Each entry has a timestamp defining the time when the state of the gates should change to a given state.
4) Transmission selection: in case multiple gates are open at the time, this entity will refer to a second scheduling algorithm (e.g. credit based shaping) to decide which packet should be transmitted first.

IV. Scheduling & Routing In SD-TSN

In SD-TSN, we combine both coarse-grained schedules (determine the transmission time at the end devices) and fine-grained link schedules (determine the transmission time at the network core). In the following, we present the main approach for computing transmission schedules and routing for SD-TSN, which is based on rapidly scheduling TSN flows incrementally using network and application contracts, thereby reducing the execution times, while maintain the quality of the schedules in terms of number of accommodated flows.

A. Scheduleing approach in SD-TSN

For each TSN flow, end devices(hosts) sign a contract with the TSN controller by providing information about the flow bandwidth [e.g. the maximum packet size and maximum packets per time interval]. In return, the TSN controller reserves bandwidth, buffering, and scheduling resources for this TSN flow. The signed contract provides the following aspects:

- Allocating enough buffer space for the TSN flow to minimize congestion loss.
- Pacing the TSN flow across the network [speeding or slowing] to deliver packets just-in-time, rather than as soon as possible for better network determinism.
- Allow non-TSN traffic to use any non-contracted bandwidth by TSN flows.

In addition, contracts are flexible, by means new contracts can be made, and old ones can be removed. Transmission schedules are modelled in a cyclic fashion, each cycle is divided in to N slots, and has duration equivalent to base time t_{base}. Each slot duration $t_{slot}$is bounded by the longest network path latency $t_{longest\_path}$.

- Number of slots N based on t_base and t_slot, both are calculated by the network controller as system parameters

$$N = INT[\frac{t_{base}}{t_{slot}}], t_{slot} \leq t_{longest\_path} \quad (1)$$

- TSN controller will allocate a slot for each contracted TSN flow. Thus, it will form the GCLs to be distributed among the network devices (TSN switches). Then, each TSN switch will disperse the exact allotted slot to the sources of TSN flows.
- All the devices have synchronized clocks [IEEE 802.1AS]. Therefore, TSN switches will refer to the same cycle base time with their schedules, while end devices (hosts) will compute the exact transmission instants using the base-period, the slot length, its transmission period, and the allocated time-slot.
- Sources transmit TSN packets that belongs to a TSN flow at the exact allotted transmission time. However, if they have nothing to transmit then they either transmit nothing or a non-TSN packets (e.g. web traffic).

This approach benefits the network utilization. Especially, with the network ability to carry best effort traffic while waiting for TSN packets to arrive during a time-slot. In addition, TSN switches could be equipped with frame preemption mechanisms [IEEE 8021.Qbu] to preempt best-effort frame to handle the higher priority frame first.

B. Routes calculation in SD-TSN

The network controller calculates a set of shortest paths, to route the TSN flows through, at first when end devices sign a contract to reserve a path for their traffic. Therefore, each TSN flow is restricted to be routed through one of the paths in this set rather than exploring all other random paths. Calculating the optimum paths is a complex problem however the overall runtime can be reduced by searching in a smaller set. To formulate this problem we assume:

- Set of free time slots that can be assigned to a path, T. $T \equiv \{t_l\}, l\epsilon[1, \dots L]$, where L represents the number of free slots that can be assigned to a path.
- Set of TSN flows for an end device, F. $F \equiv \{f_i\}, i\epsilon[1, \dots NF]$, where NF represents the number of flows to be scheduled.
- Set of candidate paths for a TSN flow, CP. $CP \equiv \{cp_k\}, k\epsilon[1, \dots NCP]$, where NCP represents the number of candidate paths between source and destination for each flow $f_i \epsilon F$.
- Each $f_i \epsilon F$ maps to $cp_k \epsilon CP$, $F_C P$. $F_{CP} \equiv \{f_{cp}^{i,k}\}; \forall i \epsilon NF, \forall k \epsilon NCP$, where $f_{cp}^{i,k}$ indicates the possibility of routing flow i, f_i, over candidate path k, cp_k.
- Each $cp_k \epsilon CP$ maps to $t_l \epsilon TL$ to, to CP_T. $CP_T \equiv cp_t^{k,l}$; $\forall k \epsilon NCP, \forall l \epsilon L$, where $cp_t^{k,l}$ indicates the allocated free slot to a certain path in order to carry the TSN flow.

The objective function is to maximize the number of flows that could be routed through candidate paths with allocated time slots.

$$maximize \sum_{\forall i \epsilon NF} [\sum_{\forall k \epsilon NCP} [f_{cp}^{i,k} * \sum_{\forall l \epsilon L} [cp_t^{k,l}]]] \quad (2)$$

Considering these constraints:

- At most one slot allocated to a candidate path.
- Paths with shared links must not be allocated the same time slot.

Finally, the network controller allocates time slots to the candidate path based on flow rate and the timing requirements to guarantee end to end latency less than $t_{longest\_path}$ and therefore routes the TSN flow packets through the assigned candidate path.

V. Validation and Discussion

In this section, we detail how we have conducted the evaluation and have tested our main contributions. We first describe the test environment that we use to evaluate the performance of SD-TSN and then we proceed with discussing the experiments and the achieved results.

A. SD-TSN test environment setup

The test environment is based on integrating several open-source software tools, which can be summarized as follows:

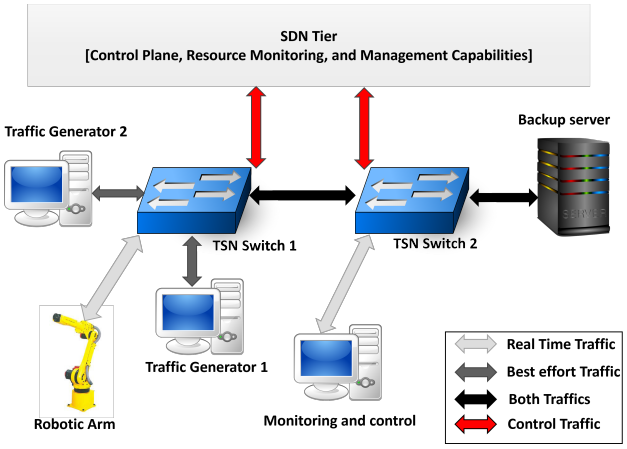

Fig. 4. SD-TSN testing use case.

- Mininet: a common tool for emulating SDN-based networks. We use it to emulate end devices and network switches. Thus, we can construct virtual networks, running a real kernel, on a single machine to test different scenarios based on our SD-TSN architecture.
- TSN software switch: based on a software Open vSwitch (OVS), the TSN switch supports both the sFlow protocol, which is used for monitoring network traffic, and the OVSDB protocol, which is used to configure the TSN switch according to the protocol processed messages sent by the network controller. The switch's kernel is modified to support multiple queues (mq) and to configure the queuing discipline (Qdisc) as Time Aware Priority Shaping technique (TAPRIO Qdisc), which allows the configuration of the GCL of the switch output ports.
- sFlow collector: continuously receives sFlow datagrams from all the network devices. It translates those datagrams into metrics that provide information about flows, communication interfaces and the status of the devices.
- Network controllers: two controllers, network OpenFlow controller and TSN controller, to provide a fully functional control plane responsible for different tasks described earlier in Section III.

Moreover, a software implementation for the objective function and the solver to maximize the number of routed TSN flows within a determined bounded latency have been made.

B. SD-TSN emulation condition

Figure 4 shows a testing use case for a network that implements the SD-TSN architecture. The use case includes different network name-spaces that represent network devices (TSN switches, TSN controller, and Network controller), and host name-spaces that mimic the behavior of real time and best effort traffic sources (Traffic generators, Robotic Arm, Monitoring and control unit, and Back up server). This is in addition to virtual Ethernet pairs, which allow the name-spaces

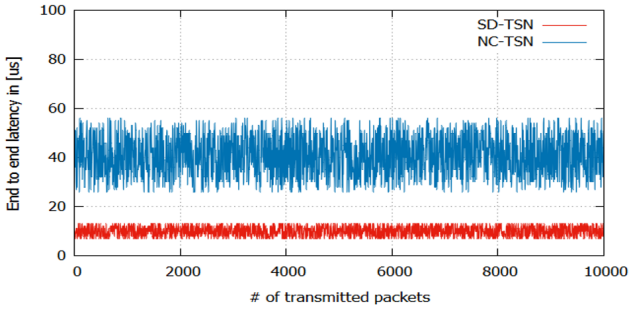

Fig. 5. End to end latency versus number of real time transmitted packets.

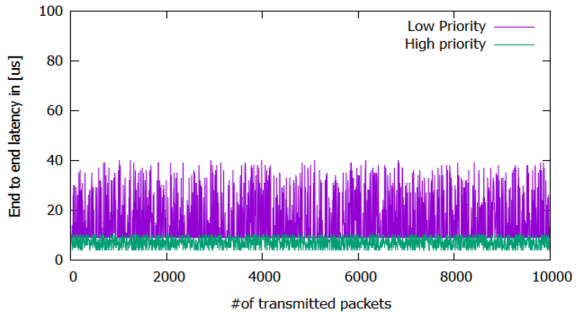

Fig. 6. End to end latency for different types of traffics with different priorities.

to communicate between each other or with the main host name-space. We use Linux Traffic Control and NetEm tools [16] to configure the emulation parameters, which allows us to emulate various capabilities of links and set different network conditions, such as:

- configuring link parameters, packet queues on interfaces, and bandwidth constraints.
- Emulate packet loss rate to characterize path performance degradation.

Tools such as ping, iperf or Netperf, have been used to generate ICMP, TCP and UDP flows. Those flows are mapped into:

- Real time traffic: which represents the TSN flows need to be sent across the network and delivered at a certain time.
- Best effort traffic: which represents non-TSN flows for example web traffic or data streams across the network.
- Control traffic: which represents the traffic flow between network devices to configure and maintain the network operation.

The emulation process allows to check the proper operation of the network design and validates the traffic scheduling aspects. Moreover, as Mininet provides a Python API to define arbitrary topologies, different scenarios are defined in accordance with the specific features and metrics being tested.

C. SD-TSN performance evaluation

While SD-TSN adheres to the computed schedules as outlined in section IV.B, non-constrained TSN (NC-TSN) requires the network controller to search the complete solution space for available paths. Thus, SD-TSN adds more determinism to the network. Therefore, in order to properly assess the performance of SD-TSN compared to the basic NC-TSN, we focus our experiments on measuring the end to end latency for TSN flows for different scenarios. Thus, we can evaluate the performance of SD-TSN in providing hard real-time guarantees to support industrial environments. In addition, we assess the capability of the SD-TSN control plane in computing accurate, correct, and scalable transmission schedules.

In Figure 5, we assess the end to end latency for 10,000 packets (real time traffic) sent across the network with a period of 20 ms. For NC-TSN, the results show the latency varying between 27– 56 µs, with an average latency of 43.46 µs and a standard deviation of 5.32 µs. This happens due to variable delays (20–100 µs) that packets incur while traversing through the network stacks in addition to those packets that are not assigned precisely to a network interface with deterministic delay. However, for SD-TSN, the results show the latency varying between 7– 13 µs, with an average latency of 9.64 µs and a standard deviation of 2.32 µs. The low end-to-end latency indicates that packets are accurately placed on the network interface with the minimum deterministic delay. Thus, SD-TSN adheres to the transmission schedules with high accuracy, in addition to providing tight bounds on the delays incurred in the end systems.

SD-TSN is designed not only to handle real time traffic but best effort traffic as well. Therefore, we conduct another experiment where we assess the end to end latency for 10,000 packets of a mixed traffic (real time and best effort) sent across the network with a period of 20 ms. The TSN real time traffic is assigned high priority to meet the real time requirements while best effort traffic is assigned lower priority. Figure 6 depicts the end-to-end latency for the TSN traffic varying between 7-13 µs, which did not change as TSN switches support frame preemption, which means higher priority frames will not be affected by the lower priority best effort traffic. However, the best effort traffic will be affected due to prioritization, yet it achieves latency varying between 9-41 µs which is acceptable for having better network utilization.

Finally, we assess the scalability of SD-TSN versus the basic NC-TSN through increasing the number of scheduled TSN flows across the network. For this experiment, we used a tree topology of 4 TSN switches and 20 end devices, in addition to the network and TSN controller. The objective function is to maximize the number of flows that could be routed with allocated time slots. We measure the run-time for computing the accurate schedules with a varying number (10–160) of flows. Figure 7 shows that the run-time for computing the schedules using SD-TSN is at least an order of magnitude lower than that for computing it using NC-TSN. It takes SD-TSN 10 seconds (100 ms per flow * 100 flow) to compute schedules for over 100 flows. While computing the schedules, for the

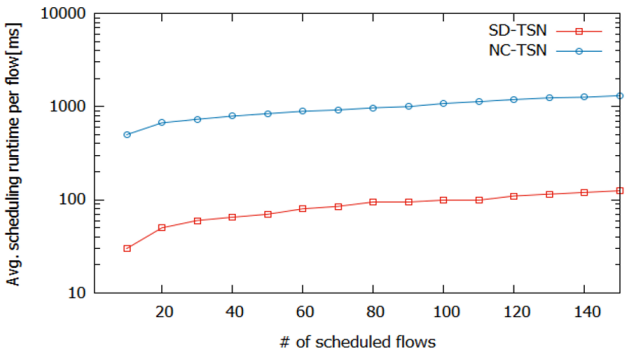

Fig. 7. Average scheduling run time per TSN flow versus number of scheduled flows across the network.

same number of flows, using NC-TSN requires approximately 2 minutes. In summary, the evaluation results show that: 1) SD-TSN guarantees an end-to- end latency less than 13 µs for scheduled TSN flows; 2) Additionally, supporting best effort traffic without affecting the latency of the real time traffic and achieving better network utilization; 3) SD-TSN scales well in terms of supporting more than 100 flows in approximately 10 seconds for a medium size network.

VI. Conclusion and Future Work

In this paper, we discuss the feasibility of using software defined network concepts in conjunction with time sensitive networking to achieve greater levels of network resource management, while meeting QoS requirement with high reliability, specifically for time-critical industrial network systems. We present Software-Defined Time Sensitive Networks (SD-TSN), which provides real time guarantees for time sensitive traffic while providing better network utilization through supporting other types of traffic. Starting from a description of the factory model and how it will be mapped to a designed architecture, we formulate an approach to compute transmission schedules. We implement a prototype of our architecture in Mininet with supporting software tools. Our evaluations showed that our SD-TSN scheduling approach calculates accurate transmission schedules which assures bounded latency for TSN flows, in addition to supporting a large number of flows with low runtime [10 seconds for 100 flows]. In our future work, we will be focusing on 2 main points: First, we plan to extend our model to support wireless devices where we can use SD-TSN as backbone for larger industrial networks. Secondly, we will use accurate models of end devices and channel models to test our design for reliable results.

References

[1] R. Drath and A. Horch, “Industrie 4.0: Hit or Hype?” Industrial Electronics Magazine, IEEE, vol. 8, no. 2, pp. 56–58, Jun. 2014. Available: http://dx.doi.org/ 10.1109/mie.2014.2312079

[2] T. Sauter, “The Three Generations of Field-Level Networks - Evolution and Compatibility Issues,” Industrial Electronics, IEEE Transactions on, vol. 57, no. 11, pp. 3585–3595, Nov. 2010. Available: http://dx.doi.org/10.1109/TIE.2010.2062473

[3] B. Galloway and G. Hancke, “Introduction to Industrial Control Networks,” Communications Surveys Tutorials, IEEE, vol. 15, no. 2, pp. 860–880, May 2013. Available: http://dx.doi.org/10.1109/SURV.2012.071812.00124

[4] Kim, Ikhwan, and Taehyoun Kim. ”Guaranteeing isochronous control of networked motion control systems using phase offset adjustment.” Sensors 15, no. 6 (2015): 13945-13965. Available: https://pubmed.ncbi.nlm.nih.gov/26076407/

[5] M. Felser, ”Real-Time Ethernet - Industry Prospective,” in Proceedings of the IEEE, vol. 93, no. 6, pp. 1118-1129, June 2005, Available: http://dx.doi.org/ 10.1109/JPROC.2005.849720.

[6] Mininet, An Instant Virtual Network on your Laptop (or other PC) Available: http://mininet.org/

[7] Deterministic Networking Charter. Available: https://datatracker.ietf.org/wg/detnet/charter/.

[8] Danielis, Peter, Jan Skodzik, Vlado Altmann, Eike Bjoern Schweissguth, Frank Golatowski, Dirk Timmermann, and Joerg Schacht. ”Survey on real-time communication via ethernet in industrial automation environments.” In Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), pp. 1-8. IEEE, 2014.

[9] Kálmán, György. “Mass Configuration of Network Devices in Industrial Environments.” . In The Twelfth International Conference on Networks (ICN 2013), pages 107–111. IEEE (2013).

[10] M. Herlich, J. L. Du, F. Sch¨orghofer, and P. Dorfinger. Proof-of-concept for a software-defined real-time ethernet. In Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on, pages 1–4. IEEE, 2016.

[11] Said, Siwar Ben Hadj, Quang Huy Truong, and Michael Boc. ”Sdn-based configuration solution for ieee 802.1 time sensitive networking (tsn).” ACM SIGBED Review 16, no. 1 (2019): 27-32.

[12] Nayak, Naresh Ganesh, Frank Dürr, and Kurt Rothermel. ”Time-sensitive software-defined network (TSSDN) for real-time applications.” In Proceedings of the 24th International Conference on Real-Time Networks and Systems, pp. 193-202. 2016.

[13] W. Steiner, ”An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks,” 2010 31st IEEE Real-Time Systems Symposium, San Diego, CA, 2010, pp. 375-384, doi: 10.1109/RTSS.2010.25.

[14] Casado, Martin, Keith Eric Amidon, Peter J. Balland III, Natasha Gude, Justin Pettit, Benjamin Levy Pfaff, Scott J. Shenker, and Daniel J. Wendlandt. ”Network operating system for managing and securing networks.” U.S. Patent 9,083,609, issued July 14, 2015.

[15] N. McKeown, T. Anderson, et al. Openflow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2):69–74, 2008.

[16] S. Hemminger, “Network emulation with NetEm,” in Australia’s National Linux Conference (LCA), 2005, pp. 18–28.