Residual Service Time Optimization for legacy Wireless-TSN end nodes

Pablo Avila-Campos, Jetmir Haxhibeqiri, Merkebu Girmay, Ingrid Moerman, and Jeroen Hoebeke
IDLab, Department of Information Technology at Ghent University - imec, 9052 Ghent, Belgium
{pabloesteban.avilacampos, jetmir.haxhibeqiri, merkebutekaw.girmay, ingrid.moerman, jeroen.hoebeke}@ugent.be

***Abstract*—The emergence of Time-Sensitive Networking (TSN) has enabled network determinism to a new level, offering high reliability and bounded latency for critical communications. However, the unpredictable nature of traffic generation also poses new challenges to TSN. While TSN is designed to maintain backward compatibility with the 802.1 standards, many end nodes may not be equipped to understand TSN. This can result in a less deterministic TSN, and suboptimal resource utilization, mainly driven by Residual Service Time (RST). To address these challenges, this study proposes three scheduling mechanisms to reduce RST: q-learning, active time slot update, and polynomial forecasting. Real-world data captured from our wireless-TSN (W-TSN) evaluation kit is used to compare the proposed approaches in terms of one-way latency. The results show that the machine learning approach outperforms the other methods in terms of overall latency. However, it is less effective in identifying the optimal time slot position compared to the other methods.**

***Keywords*—wireless time-sensitive networking; scheduling; machine learning; reinforcement learning; residual service time.**

I. Introduction

Led by the promise of bounded latencies and guaranteed packet delivery, Time-Sensitive Networking (TSN) is attracting the attention of industry [1], [2]. New paradigms that mingle digital and physical worlds such as the metaverse are grounded by the integration of extended reality (XR) and degrees of augmented and virtual reality (AR/VR). This integration is opening a new window of applications that include basic tasks such as virtual or real objects' movement, where end-to-end communication latency of less than 1 ms and 99.999% reliability are a must to enjoy a smooth experience [3].

Furthermore, such network demands are also required by mission-critical industrial applications such as aerospace, automotive, and machine control applications. Most of these applications currently depend on the old-fashioned best-effort network packet delivery capabilities, which in most cases are unacceptable [4]. Additionally, for a wide range of these applications having a wired solution is not sufficient. Hence, wireless-TSN (W-TSN) as an extension of TSN, which would bring mobility and plug-and-play capabilities, represents a natural next move [5].

TSN's strategy to achieve an advantage over traditional networks is based on combining features leading to determinism. Such components are defined by the IEEE TSN Task Group[1] in four categories: Synchronization, Reliability, Latency, and Resource Management. Every component comprises a series of standards, some of which are still under development [6]. However, even though TSN is developed over IEEE 802.1, during the initial deployment stage, it is expected that most end devices or data generators will not be fully TSN compatible. As a result, some of its benefits will only be partially available to those non-TSN end devices.

Jitter, a widely known concept in best-effort networks, is highly reduced in TSN thanks to transmission time slot alignment achieved by the IEEE802.1Qbv Time-Aware Scheduler (TAS) [7]. Hence, a frame would hop between nodes toward its destination with a reduced delay. However, the incorporation of non-TSN end nodes in a TSN might still generate latency jitter that is proportional to Residual Service Time (RST). The RST comes as an intrinsic characteristic of the cyclic scheduling in TSN and is defined as the elapsed time when a frame is available to be transmitted until it gets transmitted in the medium. The addition of an end node in a TSN is typically achieved by connecting it directly to a TSN switch for wired connections, or by using a W-TSN station (STA) for wireless connections [8]. This represents a key difference between the two types of TSN, and provides an opportunity to optimize the RST through the use of the STA.

RST mainly occurs when non-TSN end devices generate frames independently of the TSN time-synchronization or scheduling process. This behavior can be acceptable for some applications that primarily require reliability, but it is not suitable for most time-sensitive traffic flows, as it leads to suboptimal use of network resources and non-deterministic waiting times. Given that the problem is mainly related to the lack of control over frame generation, it is natural to propose approaches that focus on two main aspects: i) incorporating TSN features into end nodes, or ii) implementing forecasting-related solutions such as prediction algorithms or machine learning techniques for frame generation prediction. The latter being the specific focus of the present work.

Based on real captured data using our imec's W-TSN evaluation kit which has been built on top of openwifi [9], this paper's main contribution corresponds to a one-way latency comparison of an optimization best-guess-based selection algorithm, and a polynomial-fitting forecasting method, against a reinforcement learning approach for non-TSN end-device frame generation prediction. An efficient wireless STA-based generation forecast capable of processing small time slots could align TSN time slots with data generation time, thus

[1] http://www.ieee802.org/1/tsn

reducing latency and jitter in the TSN.

The remainder of the paper is organized as follows. First, in Section II, related works on TSN residual service time are presented, followed by a brief overview of residual time optimization in Section III. Next, in Section IV, the proposed management algorithms are explained, to finally present the results in Section V and conclude in Section VI.

II. Related Works

The absence of a standardized TSN traffic scheduling algorithm has resulted in a substantial amount of work from the community. This work is mainly concentrated on network calculus, machine learning, or heuristic optimization approaches that tackle the problem from a system-wide perspective. As a result, most of the defined systems consist of a static collection of interconnected TSN switches with end nodes as flow generators [8], [10], [11], [12].

Some studies, such as the one in [13], take an additional step by addressing end nodes without synchronization or scheduling to provide timing guarantees. This approach aims to reduce the likelihood of non-TSN and TSN traffic arriving at the same priority switch queue, thereby affecting TSN traffic determinism. However, our work differs significantly from these studies in several key ways. Firstly, we concentrate solely on the TSN access time slot (see Fig. 2), which is crucial in our proposal to integrate wireless into TSN, given the substantial differences between wired and wireless networks. In principle, we anticipate that the W-TSN will be situated at the TSN's edge, acting as access for end nodes. Therefore, unlike wired TSN, where a switch serves as the network entrance point for end nodes, in wireless, an end node's flow access device is an STA, which presents a promising opportunity point to $filter$ non-TSN traffic by implementing the strategies outlined in our research.

Moreover, authors in [14] address unsynchronized traffic by incorporating it into their mapping procedure from legacy Ethernet to wired TSN. However, they only provide a basic level of timing guarantee by classifying it as an Audio-Video-Bridging (AVB) type. Hence, lowering the TSN guarantees.

Additionally, authors in [15], propose utilizing a hierarchical structure to reduce the size of the schedule calculation problem by partitioning the stream set. However, this approach, which is commonly found in the current literature, does not consider non-TSN-compliant unsynchronized end node traffic. Nonetheless, studies like this complement the work we propose, as they would determine the best schedule from the wireless access point to the information receiver.

In conclusion, since the TSN working group began proposing initial concepts for incorporating determinism in best-effort networks in 2012, numerous studies have focused on achieving optimal scheduling. However, many of these studies do not take into account real-world hardware in their analyses, nor do they address the difficulties involved in integrating wireless into TSN.

III. Residual Service Time Optimization

This section's aim is to provide an overview of RST as well as a description of the proposed strategies.

A. Residual Service Time (RST)

General strategies are defined by TSN's current standards to describe an automated traffic flow scheduler. The lack of a standardized scheduling method is not an obstacle for wired TSN due to its steady nature, where fixed schedules are manually set at the boot-up time using the Central Network Controller (CNC) and Centralized User Configuration (CUC). However, driven by the mobile nature of W-TSN the need for an adaptable scheduler is evident. Furthermore, once the schedule is computed, and distributed to the TSN end nodes, the management system will need to keep adapting to the network and end node changes. One of the most critical delays that might be improved by adaptation is the Residual Service Time (RST).

RST is mainly generated by the lack of time synchronization, variable processing times, and lack of awareness of the schedule between the end node and the TSN. In the first update stage from non-TSN to TSN, it is expected that most end devices will not have TSN capabilities. In some cases, modifying the end node software and/or hardware to include basic TSN features such as time-synchronization will not be an option. An example of the one-way latency of a non-TSN-aware end node can be found in Fig. 1.

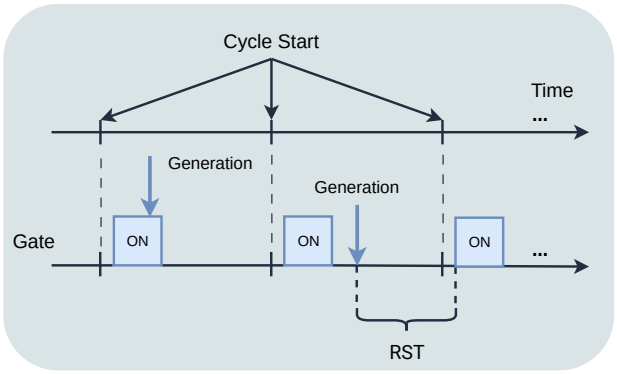

Fig. 1. RST problem

Fig. 1 shows a typical time-triggered traffic (TT) end node generation with respect to the TSN gated schedule. Despite the end node running an application with the same generation cycle as the TSN, the generation of frames is shifted due to the jitter on generation and processing time. This synchronization mismatch leads to latency jitter that could be as high as communication cycle time, an unwanted characteristic for some applications. For instance, if one transmission slot is open per cycle for the end node, the frame latency in one hop can be calculated by adding the RST, the processing delay, and the packet transmission time. Here, because of its shifting nature, the RST amount will fluctuate from 0 when both generation and TSN cycle are aligned and the generation happens during

the assigned time slot, to $Cycle - Timeslot_{size}$, when the frame is generated just after the time slot ends.

B. Strategies to Reduce RST

Theoretically, in an optimal TSN, where even processing delay is predictable (e.g., in a real-time operating system), RST is low or nonexistent. The strategies to solve or improve the RST in real development will be mainly related to the nature of the end node. There are two main aspects to consider: the traffic type and the end node capabilities to include TSN features.

Regarding the traffic type, a periodic traffic type such as the machine control one, not only reduces the mismatch possibilities but also opens the door to several strategies to reduce RST. The one proposed in this work is to provide a pulse-based time synchronization from the TSN to the end node. This would undoubtedly lower RST, however, the quality of the time synchronization, plus variable processing times at the end node would still generate RST. Hence, extra strategies need to be explored. The details behind the pulse-based synchronization are explored in section V-A. Another potential solution to synchronization would be to incorporate a packet-based synchronization mechanism like Precision Time Protocol (PTP). However, it's important to note that this approach would require the end node to be compatible with PTP in order to be effective.

The proposed strategy in this paper is to adapt the TSN access time slot position in presence of pulse-based synchronized traffic generation to reduce RST. Such adaptation, from a management point of view, has two stages. The first, as it is depicted in Fig. 2, considers the W-TSN STA node sending information such as frames arrival timestamp to the network management (CUC and CNC). By using this, and conditioned by the available resources, the time slot location can be predicted and set in an optimal place. During the second stage, the end node application has the option to delay its transmissions in order to optimize frame generation and time slot alignment. This strategy can be particularly useful when a time slot is already in use or when the ideal time slot position falls between two existing time slot locations. By strategically delaying generations, the end node can improve the RST.

IV. Algorithms Implementation

Three access time slot scheduling approaches are proposed to diminish the RST. The first is an optimization-based method that considers only the last arrival timestamp at the W-TSN STA as the best prediction for the next time slot access position. The second and third are forecasting methods. The former fits a polynomial equation using a number of past samples. In contrast, the latter uses Reinforcement Learning (RL) to model the arrivals pattern to find the best next time slot position.

As shown in Fig. 3, the scheduling works as a feedback system using the one-way latency of the frames as the algorithm's input. The output is then represented as an access time slot position.

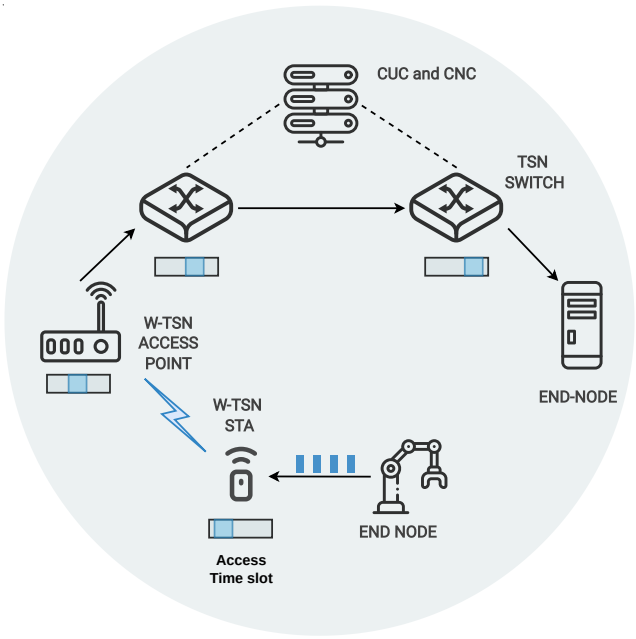

Fig. 2. W-TSN Topology

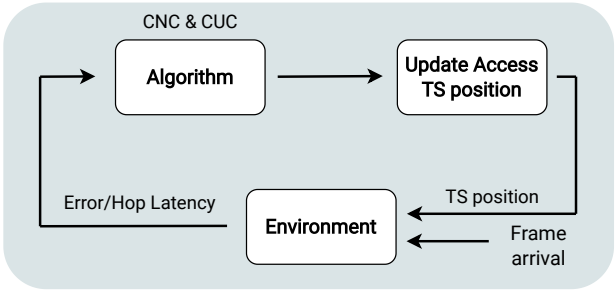

Fig. 3. RST algorithms process

Time slot length is expressed as a power of 2. As such the start and the end of the time slot can be set only on a fixed offset during the communication cycle. Hence, based on the cycle length and time slot size, there is a finite number of positions where such time slot can be placed inside the communication cycle.

The environment function task is to provide a realistic one-way latency value which is then used as a variable to optimize the algorithms. By using the cycle length, time slot position, frame arrival time, frame length, and data rate, it delivers the one-way latency of the frame. With the aim to improve the realism of the results, a random Clear Channel Assessment (CCA) delay value is also added to emulate Wi-Fi's backoff.

A. Active Update

This optimization-based function is shown in Algorithm 1. It starts by calculating an $error$, which is defined as the time distance between the previous arrival time and the middle of the time slot. Then, as in a typical proportional controller, by considering the maximum latency, the algorithm defines how many time slot positions ($steps_{left}$) the time slot should

be shifted. In addition, if a time slot is already in use, the algorithm will select the next best available time slot.

Algorithm 1 Active Update

Input: $environment(env)$, *time slot space(S)*, tx_time, TS_{size}, $cycle$, $arrivals$
Output: Access Time slot Position (s)

$error_{max} = (2 * cycle - TS\_size)/2$
for *t*=1,2,...T **do**
  $l_{now} = env(arrivals(t), s)$
  if $l_{now} > tx\_time$ **then**
    TS = S(s)
    error = $(2 * l_{now} + TS_{end} - TS_{start})/2$
    $steps_{left} = round(error * s_{max})/error_{max}$
    if $s - steps_{left} \leq 0$ **then**
      $s = s_{max} + s - steps_{left}$
    else
      $s = s - steps_{left}$
    end if
  end if
end for
return s

B. Polynomial Fitting Forecasting

The second approach is the polynomial regression forecasting which can be found in Algorithm 2. By using the last k arrivals saved in a moving vector (mv), a degree m polynomial is fitted, and the next time slot position is forecasted. Then, the best time slot position is found from the time slot position space S. The present method brings a clear advantage over the low variability of consecutive arrivals.

Algorithm 2 Polynomial Fitting

Input: $arrivals$, *block size(k)*, $degree(m)$
Output: Access Time slot Position (s)

for *t*=1,2,...T **do**
  if $mod(arrivals(t), k) = 0$ **then**
    $mv = arrivals(0 : k)$
    $function = polyfit(mv, m)$
    $prediction = function(k + 1)$
    $s = min(abs(S - prediction))$
  end if
end for
return s

C. Reinforcement Learning (RL)

For the Reinforcement Learning (RL) algorithm, a Q-learning approach is used. First, by using the arrival timestamps the exploration and exploitation stages follow an exponential decay function. The RL online learning algorithm uses the previous experience to learn its policy which is contained in a Q-function given in (1) [15].

$$Q_{t+1}(s_t, s'_t) = (1 - \alpha)Q_t(s_t, s'_t) + \alpha[r_t + \gamma max_{s'} Q_t(s_{t+1}, s')] \quad (1)$$

where γ is the discount factor, and $\alpha \in$ [0,1] is the learning rate. In the same way as in Fig. 3, the algorithm, in this case, the so-called agent, takes a decision about the time slot position, then the environment block would provide a latency reward (l) and time slot current position (s) which are used to fill the Q-table. The Q-table is a $TSxTS$ table with TS being the number of possible time slots in the communication cycle. Then, every slot would record the reward of going from TS_i to TS_j which is used in the exploring and testing stages. The Algorithm 3, presents details regarding the learning phase.

Algorithm 3 Q-Learning

Input: *time slot space(S)*, *exploration probability (ϵ), discount factor (γ),* $learning\ rate(\alpha)$
Output: Q

Initialization :
$Q \leftarrow Q_0$
for *t*=1,2,...T **do**
  $a_t = \begin{cases} argmax_a Q(s_t, a) & \text{with probability 1-}\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
  Perform action a_t
  Check new state s_{t+1} and reward r_t using environment
  $Q \leftarrow L(Q, l_{now}, \gamma, \alpha)$
end for
return Q

V. Results

This section will introduce the architecture used for measuring as well as the algorithm's results in terms of latency and accuracy.

A. Test Setup and Capture

The testing setup is divided into two parts: i) data gathering and ii) algorithms implementation and verification.

Fig. 4 shows the data collecting topology used. The end node bears an analog infrared sensor connected to an analog-digital converter attached to a Raspberry Pi 4. The former is then connected to imec's W-TSN Evaluation Kit (EK) through the W-TSN STA using Ethernet. The W-TSN STA is a ZedBoard and the AP is a Xilinx ZC706 both using a TSN-enhanced version of openwifi [2] [9]. Finally, the network controllers (CUC and CNC), are physically placed in a central node as shown in Fig. 2.

The data capture is done at the W-TSN STA driver. Here the arrival timestamp of UDP frames generated by the end node are filtered just before getting to the W-TSN STA gating system. Such capture location allows to include generation and processing delays both in the END-NODE and W-TSN STA. An electrical pulse is generated by the W-TSN STA on a cycle basis, triggering the end node's sensor capture, analog-digital conversion, and frame generation. Further, the three algorithm approaches and environment explained in Section

[2]https://github.com/open-sdr/openwifi

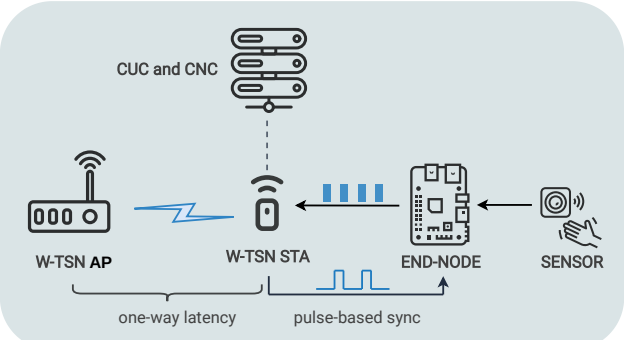

Fig. 4. Data Gathering Topology

IV were implemented in Matlab. By using the captured data set the algorithms were tested and the results are presented in the next subsection.

B. Algorithms Results

Fig. 5 presents a comparison of the mean one-way latency of the proposed algorithms using different time slot sizes for the access time slot (see Fig. 2). The results indicate that RL consistently outperforms the other algorithms in terms of mean latency across all time slot sizes. The active update approach is generally the second-best option, with the exception of the 128μs time slot, where Forecasting($k = 100, m = 3$) exhibits a lower mean latency. This poor performance of the active update approach for the 128μs time slot comes as a result of a short time slot, increasing the possibility of update cases where the time slot is positioned before the generation time increasing thus the one-way latency.

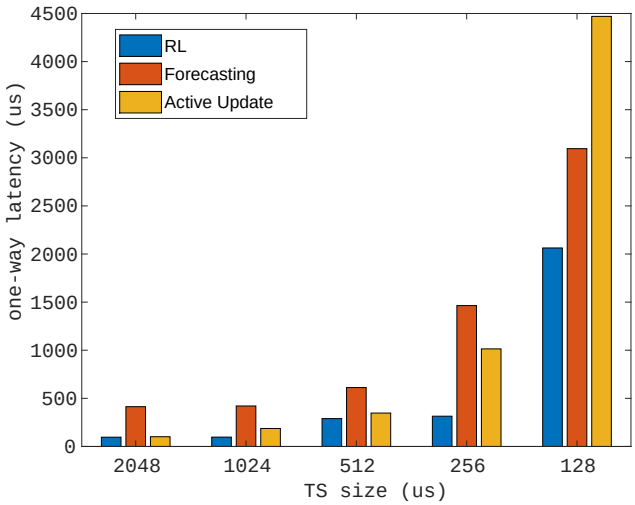

Fig. 5. Mean One-way Latency

The worst-case scenario for RST occurs when a frame is generated immediately after the time slot. To evaluate the algorithms' performance in this scenario, the percentage of frames with a latency greater than 60ms is presented in Fig. 6. The results clearly demonstrate the advantage of ML algorithms, which can learn from arrival patterns and avoid this worst-case scenario. However, the Active update and Forecasting approaches, which provide a fast response, may lead to a higher percentage of frames with latency exceeding 60ms, especially for short time slot sizes.

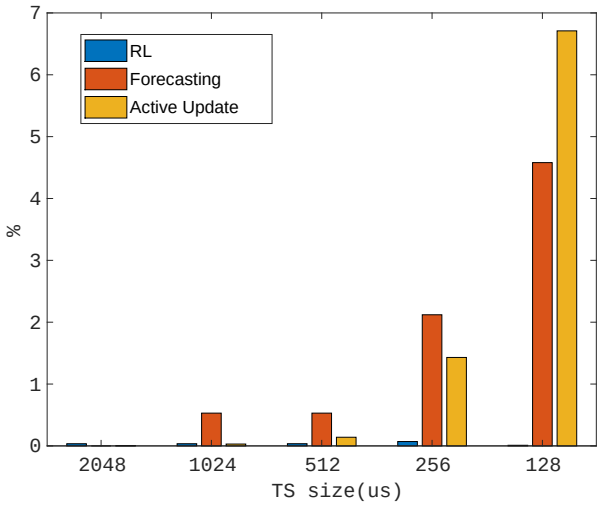

Fig. 6. Percentage of frames with $latency > 60ms$

Evaluating both the worst-case and best-case scenarios is crucial. Fig. 7 illustrates the percentage of frames with one-way latencies below 100 microseconds. The ML approach is optimized for latency, which enables it to avoid worst-case scenarios by scheduling time slots mostly after frame arrivals. However, this cautious approach may reduce the percentage of frames with the lowest latency in small time slots. Despite this, all algorithms perform well with 2048 and 1024 microsecond time slots due to the number of arrivals fitting in the time slot being bigger.

The 512μs time slot size exhibited notably poor performance. The space S of time slots is generated based on the cycle length and the number of time slots, and the algorithm selects the best possible option within this space. However, in this case, the start and end positions of the time slots are fixed, and it appears that the positions of the 512μs time slots did not contribute to accurate frame arrival times.

Data quality and quantity are crucial for ML. The Q-table size depends on the time slot; for 128μs slots, there are 262,144 positions. Filling them would improve the agent's understanding and decision-making. But only 17.87% are filled with the smallest slot. These results are not only related to the data distribution but also to the number of arrival data points used (100k).

C. Q-Table Cyclic Shifting

In the context of frame arrivals, it may be necessary to establish a fixed generation time delay at the end node in the event of a TSN schedule update. Rather than worsening the results, this fixed delay could actually help reduce hop latency by locating the frame generation in a more favorable position with respect to the time slot. However, when using Q-learning, a schedule update would alter the known information,

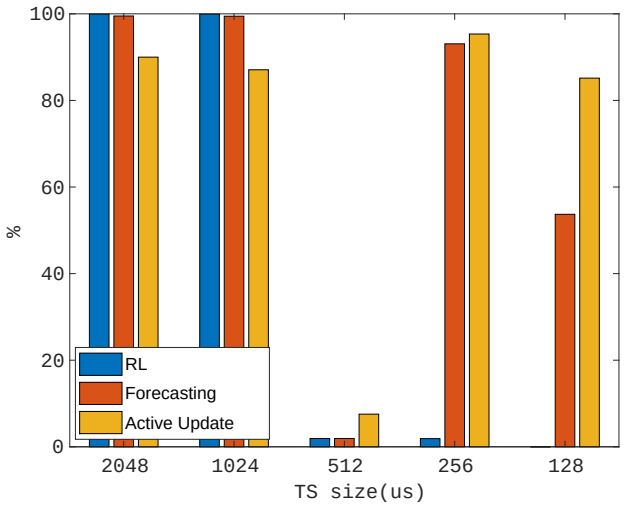

Fig. 7. Percentage of frames with $latency < 100us$

requiring a re-learning delay. To avoid this, and assuming that the distribution of frame arrivals remains unchanged, a circular shifting of the Q-table K steps is proposed to account for the fixed delay at the end node, which is denoted by (2).

$$Q_{new} = circshift(Q, K) \tag{2}$$

Table I presents a comparison of the average one-way latency between two scenarios: when training is conducted from scratch and when a circular or cyclic shift is applied to an already learned Q-table. The K-value was calculated based on the mean value of the shifted arrival distribution. Better approaches to finding K are out of the scope of this work. The table demonstrates the feasibility of using the shifting approach as an alternative to re-learning.

Table I
TRAINING VS SHIFTED MEAN ONE-WAY LATENCY

TS size (μs)	Trained (μs)	Shifted (μs)
2048	97.14	91.45
1024	96.45	96.33
512	289.46	255.61
256	314.5	308.82
128	2063	2055.66

VI. CONCLUSION AND FUTURE WORK

The Residual Service Delay (RST) poses a critical challenge in realizing the full benefits of time-sensitive networks (TSN). This study examined the problem's nature in a W-TSN context and proposed three distinct methods for reducing RST. The effectiveness of the proposed algorithms was evaluated through measuring simulations using real data frame arrival information captured from our W-TSN evaluation kit. Our findings revealed that the machine learning approach exhibited the best overall latency across different time slot sizes. However, when comparing the percentage of frames with the lowest latency in small time slot sizes, the active update and forecasting methods outperformed it. These results, coupled with the Q-table cyclic shifting proposal, represent a promising initial step towards more efficient TSN scheduling techniques. Such techniques will not only consider traffic flow requirements and resources but also constraints related to the end nodes' frame generation nature and topology in W-TSN.

ACKNOWLEDGMENT

This research was partially funded by the imec ICON project VELOCe - VErifiable, LOw-latency audio Communication (Agentschap Innoveren en Ondernemen project nr.HBC.2021.0657), the Flemish FWO SBO S003921N VERIEND.com (Verifiable and elastic end-to-end communication infrastructures for private professional environments) project, and the Flemish Government under the "Onderzoeksprogramma Artificiele Intelligentie (AI) Vlaanderen" program.

REFERENCES

[1] Cisco, "Time-Sensitive Networking: A Technical Introduction White Paper," 2017.
[2] Intel Corpotation, "Time-Sensitive Networking: From Theory to Implementation in Industrial Automation," 2018.
[3] M. Ali, F. Naeem, G. Kaddoum, S. Member, and E. Hossain, "Metaverse Communications, Networking, Security, and Applications: Research Issues, State-of-the-Art, and Future Directions," 12 2022. [Online]. Available: https://arxiv.org/abs/2212.13993v2
[4] R. Salazar, T. Godfrey, L. Winkel, N. Finn, C. Powell, B. Rolfe, and M. Seewald, "Utility Applications of Time Sensitive Networking White Paper," 2018. [Online]. Available: http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html.
[5] S. Bush, "Avnu Alliance® White Paper Wireless TSN-Definitions, Use Cases & Standards Roadmap," 2020.
[6] J. Farkas, "IEEE 802.1 Time-Sensitive Networking (TSN) Task Group (TG) Overview DetNet-TSN workshop," 2018.
[7] "IEEE 802.1Qbv-2015 - IEEE Standard for Local and Metropolitan Area Networks - Enhancements for scheduled traffic : bridges and bridged networks," p. 57, 2016.
[8] X. Liu, C. Xu, and H. Yu, "Network Calculus-based Modeling of Time Sensitive Networking Shapers for Industrial Automation Networks," *2019 11th International Conference on Wireless Communications and Signal Processing, WCSP 2019*, 10 2019.
[9] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman, "Openwifi: A free and open-source IEEE802.11 SDR implementation on SoC," *IEEE Vehicular Technology Conference*, vol. 2020-May, 5 2020.
[10] L. Zhao, P. Pop, Z. Zheng, and Q. Li, "Timing analysis of AVB traffic in TSN networks using network calculus," *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pp. 25–36, 8 2018.
[11] N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Window-Based Schedule Synthesis for Industrial IEEE 802.1Qbv TSN Networks," *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, vol. 2020-April, 4 2020.
[12] X. Wang, H. Yao, T. Mai, T. Nie, L. Zhu, and Y. Liu, "Deep Reinforcement Learning aided No-wait Flow Scheduling in Time-Sensitive Networks," in *IEEE Wireless Communications and Networking Conference, WCNC*, vol. 2022-April. Institute of Electrical and Electronics Engineers Inc., 2022, pp. 812–817.
[13] M. Barzegaran, N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Real-Time Guarantees for Critical Traffic in IEEE 802.1Qbv TSN Networks with Unscheduled and Unsynchronized End-Systems," 5 2021. [Online]. Available: https://arxiv.org/abs/2105.01641v1
[14] D. B. Mateu, M. Ashjaei, A. V. Papadopoulos, J. Proenza, and T. Nolte, "LETRA: Mapping Legacy Ethernet-Based Traffic into TSN Traffic Classes," *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 2021-September, 2021.
[15] Amrani Amine, "Q-Learning Algorithm: From Explanation to Implementation," 2020. [Online]. Available: https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187