2021 3rd International Workshop on High-Precision, Predictable, and Low-Latency Networking

Asynchronous Time-Aware Shaper for Time-Sensitive Networking

Miklós Máté, Csaba Simon, Markosz Maliosz
Budapest University of Technology and Economics
Budapest, Hungary 1117 Magyar tudósok krt. 2.
{mate,simon,maliosz}@tmit.bme.hu

***Abstract*—The Time Sensitive Networking task group of IEEE 802.1 has developed a number of enhancements to the priority queueing architecture used by Ethernet. Perhaps the most notable one is the Time-Aware Shaper, which can perfectly isolate priority classes in time with periodically scheduled traffic gates. It requires near-perfect clock synchronization across the entire network, though, and a central network controller to calculate the timings for the traffic gates. As a cheaper alternative, we developed the Asynchronous Time-Aware Shaper that needs no clock synchronization, or a central controller. It uses local processes at each switch port to track the high priority streams, predict the arrival times of their next frames, and control the traffic gates such that the low priority streams are not interfering.**

I. Introduction

Reliability in computer networks is a fundamental design aspect: the service must ensure high probability of successful packet delivery to the destinations. Several methods are used in practice to achieve reliable packet transmissions such as bit error detection and correction codes, retransmissions, and various queueing mechanisms to avoid packet loss due to congestion. Another aspect of reliability is timely arrival of mission critical packets for delay-sensitive applications. Time-Domain Multiplexing (TDM) schemes guarantee this by allocating an unique time slot for each source, thus creating contention-free access to the resources. The inflexibility of strict TDM systems made them gradually lose market share compared to networking technologies that are more universally applicable, e.g., the IEEE 802.3 Ethernet [1] for local area networks (LAN). A notable example of this trend is how circuit-switched telephony systems are phased out in favor of Voice over IP.

It has been a long time since Ethernet has abandoned its original design of randomized access to a shared coaxial cable in favor of using full duplex point-to-point links, connected by switches implementing the IEEE 802.1 bridging architecture [2]. This change effectively moved the contention for the shared resource from the medium access to the queues of the outgoing ports of the switches, where it is more manageable. This not only allowed higher link speeds, and better link utilization, but larger LAN topologies as well.

The 802.1 bridging architecture is used by not only Ethernet, but all IEEE 802 data link layer technologies, such as the 802.11 wireless LAN, promoted by the Wi-Fi Alliance [3].

Timely delivery of Ethernet frames has been in the focus of interest at the IEEE 802.1Q working group [2]. Priority queueing has been added to the bridging architecture early on, with 8 priority classes ranging from best effort to low latency voice/video traffic. Initially the 802.1Q standard only included a strict priority queueing scheme, but later amendments introduced more options for queueing and shaping.

Emerging network use cases require Ultra Reliable Ultra Low Latency (UR-ULL) communication, i.e., sub-millisecond end-to-end latency, bounded latency variation, and no packet loss due to congestion. Such use cases include industrial control loops [4], automotive applications [5], 5G wireless fronthaul [6], and tactile internet [7].

The Time-Sensitive Networking (TSN) task group of the IEEE 802.1Q working group is the successor of the Audio-Video Bridging (AVB) task group, both developing more sophisticated Quality of Service (QoS) features than strict priority transmission selection among the priority queues. They have published a number of amendments to IEEE 802.1Q standard that introduce new queueing, time synchronization, and configuration mechanisms to the 802.1Q architecture, with the aim of providing bounded latency and latency variation guarantees. The TSN task group released several amendments, in this paper we will only mention the ones relevant to our work.

An interesting new feature of TSN is frame preemption, introduced in 802.1Qbu [8] and 802.3br [9] for Ethernet, which allows the bridge to interrupt the transmission of a low priority frame to send out high priority ones that arrived in the queue. After all the high priority frames in the queue have been transmitted, the transmission of the interrupted low priority frame resumes. This feature requires major design changes in hardware; thus, only a few vendors have implemented it so far. In this paper we will assume that frame preemption is not available.

The most important TSN mechanism from our perspective is the Time-Aware Shaper (TAS) introduced in amendment 802.1Qbv [10]. It turns the priority queues into a TDM-like system governed by a global clock. In this paper we propose a modification to the TAS architecture that autonomously detects the stream parameters, and schedules the transmissions without time synchronization or centralized network management.

The rest of the paper is organized as follows. Section II gives a detailed overview of the Time-Aware Shaper. Section III

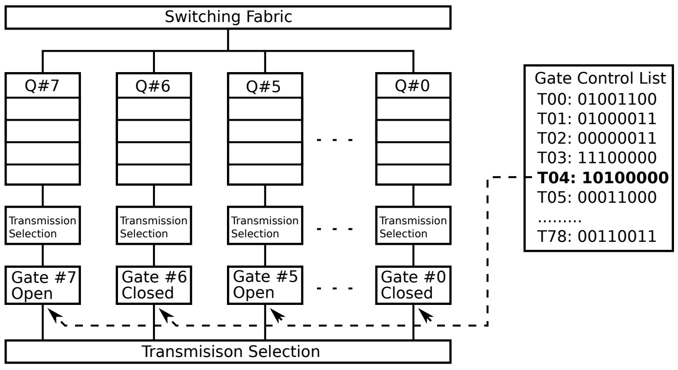

Fig. 1. Architecture of the Time-Aware Shapers in 802.1Qbv

is the main part of the paper, it presents our proposal for a mechanism to govern the Time-Aware Shaper based on local measurements and predictions. Section IV contains simulation results that show the viability of our proposal. Section V summarizes the literature related to this work. Finally, section VI draws the conclusions.

II. IEEE 802.1Qbv Time-Aware Shaper

A. *Traffic Gates*

Perhaps the most notable new queueing mechanism introduced by the IEEE 802.1Q TSN task group is the Time-Aware Shaper (TAS) added in amendment 802.1Qbv [10]. It extends the architecture of 802.1Q with transmission gates attached to each priority queue [11]. These gates have two states: open or closed. When a gate is closed, the frames waiting in the corresponding priority queue are not eligible for transmission.

The main purpose of TAS is to isolate the priority classes in time, i.e., to protect the frames of one priority class from interference by frames from other priority classes. This makes TAS a great enabler for ULL applications.

The state of the gates are governed by a Gate Control List (GCL), as shown in Fig. 1. The GCL is a sequence of 8 bit numbers corresponding to the 8 priority queues, e.g., in the figure gates #7 and #5 are open currently, while the other ones are closed. It advances with a constant tick rate, and wraps around at the end. Implementations of TAS have a finite memory capacity; thus, the GCL has a maximum available period length, depending on the tick rate, which is configurable within certain limits.

The time slice when the gate is open for a priority class is called a traffic window. The windows of the priority classes are independent, multiple ones can be open at the same time, and it is possible to close all of them. It all depends on the configuration done by the network operator. The duration of the open traffic window is also configured by the operator, e.g., a gate can be kept open all the time, it can be opened to let through only a single frame, or some duration in between.

The 802.1Qbv standard mandates that a frame is only eligible for passing through the gate, if its transmission can conclude before the traffic window closes. This implicates that the minimum window size should be larger than the packets expected in the corresponding priority class. Thus, the schedule dictated by the GCL cannot slice up the service time among the priority classes with infinite granularity.

B. *Configuration*

Configuring the TAS of the switch ports is done by a Central Network Controller (CNC) entity in the TSN architecture [4], [12]. The CNC must compute the GCL for each switch port in the network using its knowledge about the topology and the propagation delays in the network, and the stream registration information. The protocol for distributing the configuration is standardized in the 802.1Qcc [13] standard amendment, but the control functions and algorithms are proprietary solutions of the equipment vendors.

In networks with QoS guarantees there is a Call Admission Control (CAC) mechanism at the edge of the network, which allows only a limited amount of streams into the network to maintain bandwidth and latency guarantees. In the TSN architecture with TAS the CAC has to register not only the traffic characteristics of the streams (e.g. maximum data rate, burst length, packet size), but their timing information as well.

For correct GCL configuration the CNC has to know when the frames arrive at each switch. This necessitates high precision clock synchronization across the whole network, including the traffic sources. The TSN task group adopted a stricter version of the IEEE 1588 Precision Time Protocol (PTP) in amendment 802.1AS [14]. This protocol can provide clock synchronization with precision around 1 microsecond, if there is a high precision clock source in the network. This is good for industrial automation [4], where a 1500 byte frame takes around 123 μs to transmit on a 100 Mbps link (with overhead). In a 5G wireless fronthaul [6], however, the link speeds are 10 Gbps and beyond; thus, the frame transmission time is comparable to the accuracy of the time synchronization, and it becomes hard to match the opening and closing of the traffic gates with the arrival time of the frames.

It is not always possible to achieve perfect clock synchronization between the traffic sources and the transport network, especially if the two belong to different organizations. If the time synchronization is not sufficiently accurate, the traffic windows have to be enlarged to ensure safe passage for the frames before their traffic gates close, even if they are late. This decreases the throughput capacity for the other priority classes.

If a frame misses its dedicated transmission window, it has to wait for the next one, which can be much longer than waiting for an interfering frame transmission to finish.

If the transmission windows are not configured to have space for frames that missed their windows, one miss causes the next window to overflow with one frame, creating an avalanche of overflows. To stop this the GCL schedule must either increase the size of the transmission windows, or include drain windows, where the overflowed frames from multiple priority classes can leave the queue. Both are wasting link utilization, but the latter one also introduces unpredictable latency variation. Still, they are better than dropping the overflown frames.

We can thus conclude that great care needs to be taken when configuring the GCL.

In our previous work [15] we used simulations to identify configuration patterns for TSN that result in robust operation. When frame preemption is not available, we found that the safest approach for TAS is to split the eight priority queues into two groups: use the transmission gates only on the low priority group, and let the high priority traffic pass through at any time. This protects the high priority traffic without exposing them to the perils of time synchronization errors. With more than two priorities, the middle ones would have to be gated, and thus the timings would have to be much more precise. This 2-way split roughly corresponds to the express–preemptable split of 802.1Qbu.

III. Asynchronous Time-Aware Shaper

In the previous section we've seen that the IEEE 802.1Qbv Time-Aware Shaper is a powerful tool, capable of isolating traffic of different priority classes in time. It is quite expensive, though. It requires high precision time synchronization across the whole network, including the traffic sources, which might belong to a different entity than the transport network provider. It also requires a Central Network Controller overseeing the traffic, and computing the GCL for each port in the network. The CNC has to know detailed timing information about the traffic sources, and it must have precise measurements about propagation delays. In other words, the network has to be planned in advance for TAS.

This is too complicated, and expensive. Isn't it possible to set up TAS to protect high priority traffic without enormous investments into network infrastructure?

Our proposed solution for this problem is the *Asynchronous Time-Aware Shaper* (ATAS).

The idea is to run a local process at each switch port, which tracks all high priority streams passing through, predicts the arrival time of the next frames, and closes the traffic gates of the low priority queues before the next high priority frame arrives, protecting it from the interference. This solution doesn't need any central control, and no clock synchronizing, because the stream tracking and prediction mechanism runs on the local clock independent of the rest of the network (and the other ports of the switch).

Our gate control heuristic only closes the traffic gates of the low priority queues, and always lets the high priority traffic pass unobstructed. In our previous work [15] we've identified this pattern to be robust against uncertainties in the arrival times of the high priority frames.

We've designed the ATAS gate control heuristic to ensure unobstructed pass-through for high priority traffic. In ULL networks this is the expected quality level, e.g., frame preemption was designed to eliminate latency variations comparable to the frame transmit times.

When using the ATAS in the network, the admission control for the high priority streams can be much simpler than with normal TAS, as it doesn't have to register the timings of the streams, just their bandwidth requirements. Therefore, we can use the IEEE 802.1Qat Stream Reservation Protocol (SRP) [16] without modifications. SRP registers only the maximum frame rate, and the maximum frame size for the streams. Within the network no stream registration is needed, the ATAS automatically detects the stream properties.

The design of this shaper has three components: stream property tracking, prediction, and controlling the gates.

A. Prediction of Frame Arrivals

Data extrapolation, prediction, and forecasting algorithms have been extensively studied in the past several decades [17]. Widely used techniques, like Wiener filters, are designed to perform well in generic situations. In our case; however, we can exploit the traffic characteristics of high priority streams to design a much simpler predictor. We know that the sources that generate the high priority traffic are like industrial control loops [4], and base stations of cellular networks [6], which all work in a periodic fashion. We have also seen in the previous section that the GCL of the TAS is repeated periodically. Therefore, it's safe to assume that the input of our frame predictor is periodic: we have to estimate the period of the source, and its uncertainty due to random network delays.

The predictor has to converge fast on the stream parameters, because we want to minimize the number of unprotected frames at the start of the stream. There are performance concerns as well: we cannot afford complex computations in an Ethernet interface controller, and memory capacity is also scarce.

Our proposed frame predictor is deliberately as simple as possible. In section IV we'll show that even this simple predictor can protect the high priority traffic if it consists of periodic bursts.

1) Negative Correlation-based Predictor: This predictor is based on the observation that the additional network delay on a frame increases the inter-frame time before it, and decreases the one after it with the same amount. In the followings we give a formal description of the predictor.

The source sends frames periodically at T_0+iT, where T_0 is the start time of the stream, T is the period, and $i = [0, 1, \ldots]$ is the frame counter. The network delays accumulated until arriving at the outgoing queue of the TSN switch are n_i, which are independent, identically distributed random variables with unknown distribution. Frame i thus arrives at

$$x_i = T_0 + iT + n_i, \tag{1}$$

where T_0, T, and n_i are unknown to us. The goal of the predictor is to give an estimate for the arrival time of the $i+1$th frame as $\tilde{x}_{i+1}$.

The difference between two frame arrivals is

$$d_i = x_i - x_{i-1} = T + n_i - n_{i-1}, \tag{2}$$

where $i = [1, 2, \ldots]$. If we can estimate $\tilde{d}_{i+1}$, then we have

$$\tilde{x}_{i+1} = x_i + \tilde{d}_{i+1}. \tag{3}$$

We can approximate the period T with the average inter-frame arrival time. For the first N frames $\bar{T} = \frac{1}{N}\sum_{i=1}^{N} d_i$, and it's easy to see that $\mathbf{M}\{\bar{T}\} = T$ if the n_i transmission delays are identically distributed.

Our predictor computes

$$\tilde{d}_{i+1} = \bar{T} + (\bar{T} - d_i) = 2\bar{T} - T - n_i + n_{i-1}. \qquad (4)$$

We call this a negative correlation-based predictor, because it assumes a perfect negative correlation between the predicted inter-frame arrival and the current one.

It is easy to see that $\tilde{d}_{i+1}$ is an unbiased estimator of d_{i+1}. From (2) and (4) the prediction error is

$$e = \tilde{d}_{i+1} - d_{i+1} = 2(\bar{T} - T) + n_{i-1} - n_{i+1}. \qquad (5)$$

The expected value of the prediction error is

$$\begin{aligned}\mathbf{M}\{e\} =& \mathbf{M}\{2(\bar{T} - T) + n_{i-1} - n_{i+1}\} \qquad (6)\\ =& 2(\mathbf{M}\{\bar{T}\} - T) + \mathbf{M}\{n_{i-1}\} - \mathbf{M}\{n_{i+1}\} = 0,\end{aligned}$$

because n_i are identically distributed.

For each stream the computation of

$$\tilde{x}_{i+1} = x_i + \tilde{d}_{i+1} = x_i + 2\bar{T} - d_i \qquad (7)$$

is fast and its memory usage is negligible, because it only needs a moving average. If we used, e.g., a Wiener filter, we would need to store several dozen samples, and compute the inverse of a huge matrix [17].

This predictor has a fast lock-in: it can start giving usable predictions after two received frames, regardless of T_0, T, and n_i. From x_0 and x_1 we have d_1, which is the initial value of $\bar{T}$. The $\tilde{x}_2$ computed from these can be used to protect the third frame with the traffic gates.

2) Periodic Bursts: Not all traffic sources send individual frames periodically, it is possible that more data is generated, and a burst of frames are sent in each period.

To handle periodic bursts we need two cycles of the negative correlation-based predictor: one that tracks the time between bursts, and another that tracks the time between the frames within a burst. The latter is necessary, because we cannot expect the frames in a burst to arrive back-to-back. This can happen e.g. when the source has sent the frames of the burst back-to-back, but we operate on an aggregate network with higher link speeds and thus shorter frame transmission times. In this case unrelated frames from other streams may be mixed in between the frames of the burst, but that doesn't affect our per-stream predictor.

Predicting the burst size means deciding whether to expect more frames to come in the current burst, or a long silence until the start of the next burst. If the burst size is underestimated, the last frames of the burst will not be protected by the traffic gates. We took a safe approach by predicting the burst size to be the maximum number of frames in the last K bursts.

Inside the burst, the predictor uses equation (7) with x_i and d_i referring to the previously received frame. Between two bursts (when we've received the predicted amount of frames in the burst), the predictor uses equation (7) with x_i and d_i referring to the start of the last burst, and the inter-burst interval.

This prediction scheme automatically adjusts itself if the source changes its behavior regarding to burstiness.

B. Stream Property Tracking

Now that we know what stream parameters the predictor needs, we can design their acquisition process.

First, we need to identify the high priority streams. Amendment 802.1Qci adds Per-Stream Filtering and Policing (PSFP) capability [18]; thus, the bridges can identify streams and track their properties. We reuse this capability for the ATAS.

The data needed by the predictor are the following. When frame i arrives, we need to save x_i, and update the average inter-frame time $\bar{T}$ with $d_i = x_i - x_{i-1}$. The gate control heuristic will also need the average frame size of the stream.

In our implementation of the ATAS for the simulation study we used the Exponentially Weighted Moving Average (EWMA) algorithm to track the average inter-frame time and the average frame size in the stream. The new average is computed as $\bar{T} = \alpha d_i + (1-\alpha)\bar{T}$. This is fast, requires no extra memory, and it can adapt to changes in the behavior of the source, because it assigns diminishing weights to older values. The convergence speed, and the noise canceling strength of EWMA depend on the α parameter.

To track the bursts, first we need to detect if the stream has bursts. We implemented a simple heuristic for this: if frames start arriving periodically, then there is a large gap between two frames, and the previous inter-frame time resumes, we declare the stream as bursty. Our heuristic contains limits on the burst size and the length of the silence to distinguish between bursts, missed frames, and the stream stopping and restarting.

Once we know that the stream has bursts, we need to keep track of the time between the burst starts the same way as tracking the time between the individual frames. We also need to remember the number of frames received in the last K bursts, including the current one.

C. Gate Control Heuristic

Once the stream properties are tracked, and the arrival time of the next frame of each stream are predicted, we need an algorithm for closing and opening the traffic gates of the low priority queues. As mentioned before, the goal is to eliminate the interfering low priority traffic from the way of the high priority frames that belong to ULL applications.

The strength of the protection mostly depends on the gate control mechanism. We chose a simple heuristic as a proof-of-concept. We'll show in section IV that even this simple heuristic can efficiently protect the high priority streams from the low priority ones. It can be extended in the future to allow fair bandwidth sharing for the low priority streams.

Our gate control heuristic looks for the high priority stream with the earliest predicted frame arrival, and schedules the low priority gates to be closed at the predicted arrival time.

If a high priority frame arrives earlier than predicted, the low priority frames can block it until the predicted arrival

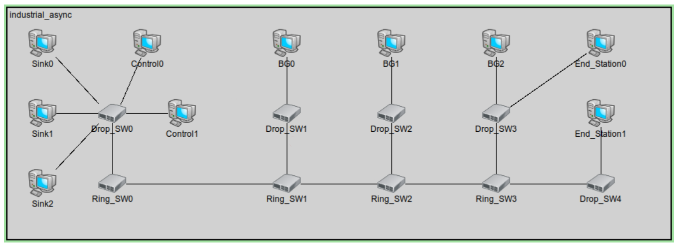

Fig. 2. The Industrial Ethernet topology used in the simulations

time, where they are definitely stopped by their transmission gate. We could alleviate this by closing the gates earlier, sacrificing some link utilization for the low priority traffic. Unfortunately, optimizing between link utilization and the probability of arriving too early is like comparing apples to oranges; therefore, we decided to not to close the gates earlier than the predicted arrival time.

The gate close times have to be calculated in advance, because the queues have to know if the low priority frames can be transmitted before their gates close.

The duration of the closed state determines the delay tolerance for the high priority frame, and the remaining time for low priority frames. If the high priority frame arrives much later than predicted, the low priority gates are open again, and now the frames that were held back interfere with it, making it even later at the next switch. In our simulations we kept the close duration at the minimum reasonable length, which is the length of the high priority frame on the wire.

It's not worth setting the close duration shorter than the time to transmit the high priority frame, because the high priority frame occupies the outgoing link anyway (we assume that it arrives as predicted). Setting longer close state could help with unexpected delays on the high priority frame, but unless we know there are such delays, we just end up wasting bandwidth. These unexpected delays can only be caused by interference from other high priority streams, as the ATAS protection is enabled on all nodes in the network.

Although this simple gate control heuristic works for any number of high priority streams, it cannot protect them from each other, because it can only distinguish between two priority levels. It may also starve the low priority traffic if the high priority frames arrive in a scattered pattern, and the open time windows for the low priority traffic become too short. Solving these problems would be hard, because using the traffic gates on the high priority streams would introduce latencies that can be unacceptable for ULL applications.

IV. Simulation Results

We examined the behavior of the ATAS with simulations in OMNeT++ to assess its viability. The OMNeT++ framework allows fast prototyping and provides several tools for state inspection and result evaluation. We reused our implementation of the TSN queueing mechanisms for the INET model set [19] of OMNeT++ that we used in our previous work [15], where we examined the robustness of TAS under various configuration patterns. The NeSTiNg model library for simulating TSN

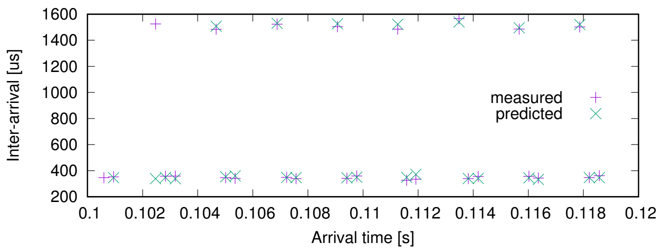

Fig. 3. Comparison of the predicted and the measured inter-arrival times

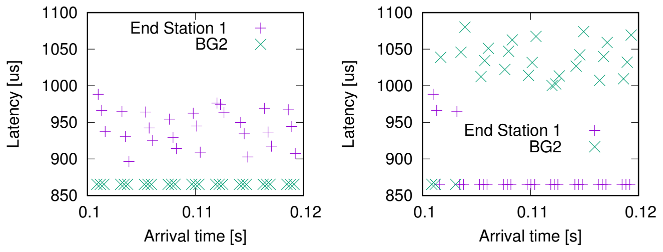

Fig. 4. End-to-end latencies with gating disabled and enabled

in OMNeT++ [20] was published not long after we were done with that project. We decided not to switch over to that model library for ATAS, because our models work almost the same way, and we were more familiar with our own code.

In the simulations we used $\alpha = 0.3$ for the EWMA algorithm, and stored the number of frames received in the last $K = 5$ bursts. These values were chosen as reasonable defaults, we'll see the effect of these parameters in Fig. 6 and Fig. 7.

The simulated network topology is shown in Fig. 2. It models a slice of an industrial control ring, with the process controller on the left receiving periodic status updates from the process monitoring nodes connected to the Drop switches dangling from the corresponding Ring switches. All links are 100 Mbps. We removed the ring nodes that are not part of the shortest path between these monitoring nodes and the controller. The End Station nodes are the sources of the high priority streams, and the BG nodes send low priority background streams. The control node is split into dedicated receiver nodes for each source to separate the streams for analysis. This topology was constructed to contain all the scenarios we need to demonstrate the capabilities of ATAS.

In a real network the source would send its frames with good accuracy, and the uncertainties would come from clock synchronization issues, and random delays in the network. In our simulations these uncertainties are added at the source; thus, when there is no interference, the end-to-end latency is constant, yet the predictors at the switches see random frame arrivals. The plots in this paper were generated with variation in the frame sending intervals of the high priority stream uniformly distributed in a 40 μs range, which corresponds to the transmission time of 500 bytes on the wire. The burst size was set to three frames for all streams.

Fig. 3 illustrates the prediction accuracy for a single high priority stream by comparing $\tilde{d}_{i+1}$ to d_{i+1}. The small inter-arrival times are between frames within a burst, while the large

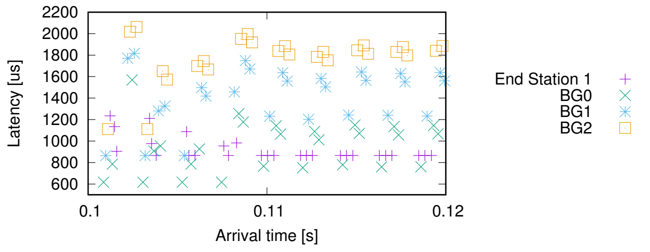

Fig. 5. End-to-end latencies with all three low priority streams enabled

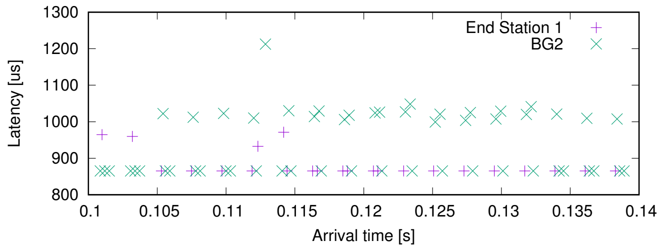

Fig. 7. End-to-end latency when the source starts and stops being bursty

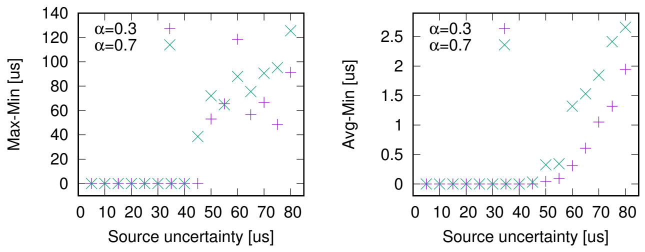

Fig. 6. End-to-end latency analysis as a function of the uncertainty of the arrival time of the high priority frame

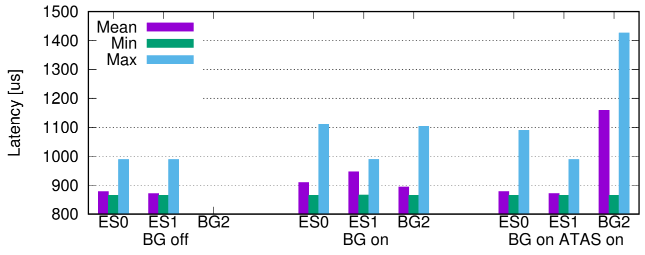

Fig. 8. End-to-end latency comparison with two high priority sources and one background

ones are the times between the bursts. The first prediction happens for the second inter-arrival time, corresponding to the third frame received, and it's quite accurate in this case. The fourth frame is incorrectly predicted, because the burstiness of the stream comes unexpected the first time it happens. The rest of the arrival times are predicted with good accuracy, even the delay pattern is matched in most cases.

Fig. 4 shows the end-to-end latency for the traffic of End Station 1 and BG2. This scenario was crafted for the worst case: the two streams have identical timings, but when the streams meet at Ring SW3 the low priority frames always arrive just before the high priority frames, as the plot on the left shows. The plot on the right shows the ATAS in action: after learning from the first few frames, the rest of the high priority stream is perfectly guarded from the interference.

Fig. 5 shows the end-to-end latencies when all three low priority streams interfere with the traffic of End Station 1. This is also a hand-crafted worst case: when Ring SW3 locks onto the stream, and starts protecting it, the decreased delay causes it to start competing with the traffic of BG1 at Ring SW2. After that competition is eliminated, the situation repeats with BG0 at Ring SW1. The learning phase gets longer with more switches along the path, but eventually all asynchronous shapers lock onto the high priority stream, and it gets protected along the whole path.

Fig. 6 shows the latencies of the frames of End Station 1 when only BG2 interferes, as a function of the uniformly distributed uncertainty of the sending interval at the source, for two EWMA α values. For low uncertainty the ATAS protects the high priority stream perfectly: the minimum and maximum latencies are the same. Beyond a threshold, however, some frames arrive too late, and cannot be protected; thus, the maximum latency increases sharply. This is a rare event, though, as seen from the relatively minor increase in the average latency. In this scenario the faster convergence of the lower EWMA α was beneficial for both the average and the maximum latency.

Fig. 7 shows how the ATAS adapts to the source changing its burstiness: it starts sending one frame per period, then at 0.11 s it switches to two frames. The shaper detects this, and recalibrates itself, which takes two periods, just like at the start of the stream. At 0.12 s the source goes back to sending one frame per burst. The shaper still keeps running its protection for the second frame of the burst for $K = 5$ periods, then sees that the maximum burst length is 1, marks the source as not bursty, and stops protecting the phantom of the second frame.

Fig. 8 shows the end-to-end latencies when both high priority sources are active, but with different periodicity. Interference between them is unavoidable, but it is a relatively rare event, as the first histogram shows: the minimum and the average latency are close to each other, which means that most frames were unobstructed. In the middle histogram the background traffic is enabled. It totally disrupts both high priority streams: the three have the same latency figure even though we have strict priority queueing. In the third histogram we see that ATAS protects both high priority streams: their average is almost the same as in the first histogram. Their maximum is higher now, because ATAS cannot protect their frames when they arrive with unexpected latency due to interfering with each other.

Note that when two high priority streams have different periodicity, the period of their combined arrival pattern can be longer than the maximum period of the GCL; thus, TAS has to hold back some of their frames with the gating.

V. Related Work

Our work is not the first one to improve upon the 802.1Qbv TAS architecture. There are several papers on enhancing the central network control, e.g., by using autoconfiguration based on preexisting knowledge about the sources [5], or developing incremental algorithms [21] for real-time computing of the new gate schedule when a stream enters or leaves the network.

There are also proposals for improving the network throughput via local optimizations to the gate schedule. In the Gate-Shrunk TAS model [22] the high priority traffic window has no predefined length. Instead, the talker sends a GS frame at the end of its burst, which signals to the bridges to close the high priority window, thus letting more low priority traffic through. Obviously, this only works for one high priority talker.

The Size-Based Queuing (SBQ) method [23] attempts to reduce the link underutilization prior to closing the traffic gate when the first frame in the queue doesn't fit. It reorders the queued frames to find one that fits. Note that reordering the frames of a stream is bad for realtime applications, and most network equipment take extra care to avoid reordering frames.

Replacing the TAS with another queueing architecture has also been proposed in the TSN task group.

Cyclic Queuing and Forwarding (CQF), formerly known as Peristaltic Shaper, has been published in amendment 802.1Qch [24]. It uses pairs of queues for each traffic class, with traffic gates opened in an alternating pattern: incoming frames are directed into the closed queue, while the open queue releases the frames it has gathered, and they switch roles periodically. This reshapes the traffic, eliminating the previously accumulated random delays. The downsides of this are the additional delay of at least one cycle, and a newly introduced pseudorandom delay due to the cycle timer being independent of the traffic periodicity. For optimal performance it is advised to synchronize the CQF shapers to a global clock.

The Asynchronous Traffic Shaper (ATS) project adds the Urgency-Based Scheduler (UBS) to the bridge architecture in amendment 802.1Qcr [25]. UBS doesn't need a global clock, because it doesn't do traffic gating. It redesigns the traffic at each hop with a per-stream token bucket scheduler, thereby regulating the burstiness of the streams. This is not a replacement of TAS (the two can be used together), rather, an improvement over the Credit-Based Scheduler (CBS) of amendment 802.1Qav by the AVB task group. UBS can increase link utilization by reducing contention [26], but it does not protect the different priority streams from each other like TAS. The additional delay introduced by the reshaping can hurt ULL applications.

VI. Conclusions and Future Work

In this paper we proposed a novel mechanism for controlling the Time-Aware Shapers of IEEE TSN bridges without central control, clock synchronization, or prior configuration. Our system tracks the properties of the high priority streams going through the bridge, predicts the arrival of the next frame of each stream, and based on the predictions it closes the traffic gates of the low priority queues to eliminate interference on the high priority streams.

We have shown with simulations the viability of our Asynchronous Time-Aware Shaper (ATAS): it can efficiently protect the high priority streams from the low priority streams, even with a simple gate control heuristic. Extending the gate control heuristic with the ability to prevent starvation of the low priority traffic will turn this into a universally applicable solution for applications that require ultra low latency, e.g., automotive networks, 5G fronthaul, and tactile internet.

References

[1] IEEE Std. 802.3-2015, "IEEE Standard for Ethernet," 2015.

[2] IEEE Std. 802.1Q-2018, "IEEE Standard for Local and Metropolitan Area Networks: Bridges and Bridged Networks," 2018.

[3] IEEE Std. 802.11-2016, "IEEE Standard for Local and Metropolitan Area Networks: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 2016.

[4] Texas Instruments, "Time-sensitive networking for industrial automation," http://www.ti.com/lit/wp/spry316a/spry316a.pdf, 2018.

[5] M. H. Farzaneh and A. Knoll, "An Ontology-based Plug-and-Play Approach for In-Vehicle Time-Sensitive Networking (TSN)," in *Proc. of IEMCON'16*, Vancouver, BC, Canada, October 2016.

[6] IEEE Std. 802.1CM, "IEEE Standard for Local and Metropolitan Area Networks – Time-Sensitive Networking for Fronthaul," 2018.

[7] G. P. Fettweis, "The Tactile Internet: Applications and Challenges," *Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, March 2014.

[8] IEEE Std. 802.1Qbu, "IEEE Standard for Bridges and Bridged Networks – Amendment: Frame Preemption," 2016.

[9] IEEE Std. 802.3br, "IEEE Standard for Ethernet – Amendment: Specification and Management Parameters for Interspersing Express Traffic," 2016.

[10] IEEE Std. 802.1Qbv, "IEEE Standard for Bridges and Bridged Networks – Amendment: Enhancements for Scheduled Traffic," 2015.

[11] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, "Extending IEEE 802.1 AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic," in *Proc. of VTC'13*, Boston, MA, USA, December 2013.

[12] Cisco, "Time-Sensitive Networking: A Technical Introduction," https://www.cisco.com/c/dam/en/us/solutions/collateral/industry-solutions/white-paper-c11-738950.pdf, 2017.

[13] IEEE Std. 802.1Qcc, "IEEE Standard for Local and Metropolitan Area Networks – Amendment: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," 2018.

[14] IEEE Std. 802.1AS, "IEEE Standard for Local and metropolitan area networks– Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks," 2011.

[15] C. Simon, M. Maliosz, and M. Máté, "Design Aspects of Low-Latency Services with Time-Sensitive Networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 48–54, June 2018.

[16] IEEE Std. 802.1Qat, "IEEE Standard for Virtual Bridged Local Area Networks – Amendment: Stream Reservation Protocol (SRP)," 2010.

[17] T. Kailath, B. Hassibi, and A. H. Sayed, *Linear Estimation*. Prentice Hall, 2000.

[18] IEEE Std. 802.1Qci, "IEEE Standard for Bridges and Bridged Networks – Amendment: Per-Stream Filtering and Policing," 2017.

[19] "INET Framework," https://inet.omnetpp.org/.

[20] 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 *Proc. of NetSys'19*, Garching b. München, Germany, March 2019.

[21] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks," *Transactions on Industrial Informatic*, vol. 14, no. 5, pp. 2066–2075, May 2018.

[22] D. Hisano, Y. Nakayama, T. Kubo, T. Shimizu, H. Nakamura, J. Terada, and A. Otaka, "Gate-Shrunk Time Aware Shaper: Low-Latency Converged Network for 5G Fronthaul and M2M Services," in *Proc. of GLOBECOMM'17*, Singapore, December 2017.

[23] F. Heilmann and G. Fohler, "Size-Based Queuing: An Approach to Improve Bandwidth Utilization in TSN Networks," in *Proc. of RTN'18*, Barcelona, Spain, July 2018.

[24] IEEE Std. 802.1Qch, "IEEE Standard for Bridges and Bridged Networks – Amendment: Cyclic Queuing and Forwarding," 2017.

[25] IEEE Std. 802.1Qcr, "IEEE Standard for Bridges and Bridged Networks – Amendment: Asynchronous Traffic Shaping," 2019.

[26] J. Prados-Garzon, L. Chinchilla-Romero, P. Ameigeiras, P. Muñoz, and J. M. Lopez-Soler, "Asynchronous Time-Sensitive Networking for Industrial Networks," in *Proc. of EuCNC'21*, Porto, Portugal, June 2021.