Software-Based Time-Aware Shaper for Time-Sensitive Networks

Yasin OGE†, Nonmember, Yuta KOBAYASHI†, Member, Takahiro YAMAURA†, Senior Member, and Tomonori MAEGAWA†, Member

SUMMARY This paper presents the design, implementation, and evaluation of a time-aware shaper, which is a traffic shaper specifically designed for IEEE 802.1Qbv-compliant time-sensitive networks. The proposed design adopts a software-based approach rather than using a dedicated custom logic chip such as an ASIC or FPGA. In particular, the proposed approach includes a run-time scheduler and a network interface card (NIC) that supports a time-based transmission scheme (i.e., launch-time feature). The run-time scheduler prefetches information (i.e., gate control entry) ahead of time from a given gate control list. With the prefetched information, the scheduler determines a launch time for each frame, and the NIC controls the time at which the transmission of each frame is started in a highly punctual manner. Evaluation results show that the proposed shaper triggers transmission of multiple time-sensitive streams at their intended timings in accordance with a given gate control list, even in the presence of high-bandwidth background traffic. Furthermore, we compare the timing accuracy of frame transmission with and without use of the launch-time feature of the NIC. Results indicate that the proposed shaper significantly reduces jitter of time-sensitive streams (to less than 0.1 μs) unlike a baseline implementation that does not use the launch-time feature.

key words: time-sensitive networking, TSN, IEEE 802.1Qbv, time-aware shaper

1. Introduction

There is increasing interest in the use of Ethernet®-based networks for real-time constrained applications. Industrial automation [1]–[3] and in-vehicle networks [4]–[6] are typical examples of applications that require real-time communication. In the field of industrial networks, for example, it is crucial to manage a wide variety of different requirements for programmable logic controllers for real-time automation networking. As for in-vehicle networks, current trends indicate that electronic control units can be connected to an increasingly large number of sensors and actuators to support advanced driver-assistance systems and future autonomous vehicles.

Although specific real-time requirements may differ from application to application, the aforementioned networks demand strict real-time behavior. More specifically, they require a reliable (i.e., minimum or zero packet loss), low-latency, and minimal-jitter solution [7]. In order to meet diverse real-time requirements, the IEEE 802.1 TSN Task Group [8] has been working on a set of standards for time-sensitive networking (TSN) technologies, based on IEEE 802.1 audio/video bridging (AVB) standards [9]. In this paper, we focus on one key feature of TSN, namely, a new traffic shaping mechanism called the “time-aware shaper” [10].

The main contributions of this paper are to implement and evaluate a traffic-shaping method, specifically designed for time-sensitive networks, by using a time-based transmission scheme. The proposed method prefetches scheduling information, and determines a launch time for each frame according to the prefetched information. The scheduling information used in this work is a gate control list which is compliant with the IEEE 802.1Qbv standard [10]. The transmission scheme of the prototype implementation is based on the launch-time feature of a network interface card (NIC) [11]. This feature enables precise specification of the time at which each Ethernet® frame should be sent from the NIC with a timing accuracy of better than 1 μs.

This paper is an extended version of the authors’ previous work [12]. While custom hardware (i.e., ASICs or FPGAs) could be used to implement a time-aware shaper compliant with IEEE 802.1Qbv, a custom hardware design generally results in lower flexibility and a higher design and implementation cost. To avoid such issues, in this paper we focus in particular on a software-based approach that does not use a dedicated custom logic chip such as ASIC or FPGA. That is, the main scope of this paper is to implement and evaluate a time-aware shaper for the IEEE 802.1Qbv standard by using a general-purpose CPU and software along with an NIC supporting the launch-time feature. This provides us with a great degree of flexibility when it comes to combining the time-aware shaper with other TSN (e.g., per-stream filtering and policing [13]) or AVB (e.g., Credit-based Shaper [14]) standards in the future.

The remainder of the paper is organized as follows. Section 2 presents a brief background on the IEEE 802.1Qbv standard. Section 3 provides basic concepts underlying the proposed approach. Section 4 presents a system configuration and evaluation scenarios. Section 5 evaluates the proposed implementation with results obtained from experiments. Section 6 compares performance of the proposed implementation and the standard Linux® network stack. Section 7 surveys related work. Finally, Sect. 8 concludes the paper with a summary of the work.
2. Background: IEEE 802.1Qbv Standard

The IEEE 802.1Qbv standard [10] specifies a set of rules for deterministic transmission in a time-sensitive network. In particular, 802.1Qbv enables intermediate nodes (i.e., switches) and end nodes (i.e., senders or talkers) to transmit each frame in a time-triggered fashion [15], [16]. For this purpose, the concept of a *gate* is introduced for each transmission queue ($Q_i$ for $i = 0, \ldots, 7$).

Figure 1, for instance, shows a typical example of an end node with a single egress port. Each gate can be in either an *open* or a *closed* state at a given time. When a gate is open, transmission is allowed for the corresponding queue. On the other hand, transmission is disallowed when a gate is closed, even if a frame is awaiting transmission in the queue.

Furthermore, each state of the gates is controlled in accordance with a predetermined periodic schedule called a *gate control list* (see Fig. 2). In other words, a gate control list specifies a schedule of time intervals for each gate indicating when gates should be open or closed for transmission. The schedule is predetermined offline to meet timing requirements for a given application or network. A gate control list includes, for example, the number of entries in the list, gate states for each entry, and a length of time interval for each entry. Interested readers are referred to [10] for further details.

The *gate driver* of Fig. 1 can read gate states from the gate control list and assign a corresponding state for each gate in a timely manner. For example, when the first time interval of the list (T00 in Fig. 2) ends at time $t = t_3$, the gate driver reads gate states of the second entry (T01). In this case, only the gate of queue $Q_3$ (i.e., Gate 7 in Fig. 1) is changed from a closed to an open state, and the other gates remain closed during the time interval of T01 (i.e., until the time $t = t_6$).

It is important to note that in order for the gate driver to operate in accordance with precise timing, an accurate reference clock is required. This means that precise time synchronization is necessary for 802.1Qbv to function as intended. For this purpose, TSN provides a synchronization mechanism, such as IEEE 802.1AS-2011 [17] or IEEE P802.1AS-Rev [18]. Given the precise time synchronization and the gate control mechanism, a time-aware shaper defined by 802.1Qbv is capable of handling time-sensitive streams which have strict real-time constraints along with best-effort traffic.

3. Implementation of the Time-Aware Shaper

In this paper, we propose a time-aware shaper for IEEE 802.1Qbv by using a time-based transmission scheme. More specifically, our prototype implementation relies on a launch-time feature of an Intel® I210 NIC [11], [19]. The launch-time feature enables us to specify the time when each Ethernet® frame should be transmitted from the egress port of the I210 NIC in a highly punctual manner (see [19] for details).

3.1 Queue Configuration

A simple block diagram for the proposed time-aware shaper is illustrated in Fig. 3. All transmission queues $Q_i$ for $i = 0, \ldots, 7$ are maintained in the host memory, and only a single frame buffer is required inside the NIC, as shown in Fig. 3. In fact, four hardware queues (from $q_0$ to $q_3$) are available for transmission inside an Intel® I210 NIC, and only two of these queues ($q_0$ and $q_1$) support time-based transmission (i.e., a launch-time feature). In our implementation, $q_0$ is used as the frame buffer inside the NIC, as shown in Fig. 3 ($q_1$, $q_2$, and $q_3$ of the I210 NIC are not used in the proposed implementation; therefore, they are not depicted in Fig. 3).

To gain direct access to $q_0$, the proposed design adopts a customized igb driver for the I210 NIC, which is provided by the OpenAvnu project [20]. The custom driver, called igb_avb, provides a unique interface (application programming interface [API]) to a user-space application in addition to a standard socket interface. The proposed implementation takes advantage of a unique API, which provides low-overhead access to the NIC. This enables low-latency transmission by avoiding the overhead incurred in the net-
work stack of an operating system.

3.2 Run-Time Scheduler

The proposed time-aware shaper implements a run-time scheduler in order to schedule a launch time for each frame stored in a transmission queue (i.e., $Q_i$ for $i = 0, \ldots, 7$) are implemented in the host memory, and the run-time scheduler determines a launch time for each frame.

![Fig. 3](image)

**Fig. 3** Simple block diagram of the proposed time-aware shaper using an Intel® I210 NIC [11]. Transmission queues ($Q_i$ for $i = 0, \ldots, 7$) are implemented in the host memory shown in Fig. 3. In fact, the run-time scheduler determines the launch time based on ft instead of ct. The scheduler monitors ft and transmission queues (from $Q_i$ down to $Q_0$) in order to determine whether any frame is available for scheduling. Let us assume that the gate control list in Fig. 2 is given, and that there is a frame in queue $Q_7$ available for transmission. The scheduler should handle two different cases in this example as follows.

**Case 1:** In the first case, $ft = t_3$ as shown in the left side of Fig. 5. Then, the scheduler estimates the required amount of time for transmitting the frame stored in queue $Q_7$ by using, for example, the frame size and the link speed. With the estimated transmission time ($t_{trans}$), the scheduler also calculates an estimated time for completion of the transmission ($t_{end} = t_3 + t_{trans}$). In this case, the transmission can be completed at a time between $t_4$ and $t_5$ (i.e., $t_4 < t_{end} < t_5$) as shown in Fig. 5. This means that there is sufficient time available for the transmission; therefore, the scheduler schedules the frame to be launched at $t_3$ (i.e., $t_{launch} = t_3$), and $ft$ is updated to $t_{end}$, as shown in the right side of Fig. 5. In brief, in the first case, the scheduler determines a launch time and updates $ft$.

**Case 2:** In the second case, $ft = t_5$ as shown in the left side of Fig. 6. Then, the scheduler estimates a transmission time ($t_{trans}$) and an estimated time for completion of the transmission ($t_{end} = t_5 + t_{trans}$) in the same way as in the first case. In the second case, the transmission could be
completed at a time between t6 and t7 (i.e., t6 < t_end < t7) as shown in Fig. 6, based on the estimation. In this case, however, the transmission is not completed within the time interval T01. Recall from the gate control list in Fig. 2 that the gate of the queue Q7 should open only in the interval T01 and should be closed in the other time intervals (i.e., T00, T02, T03). The second case in particular violates the guard band, an important concept introduced in IEEE 802.1Qbv [10]. This means that there is not enough time available for the transmission; therefore, the scheduler does not schedule the frame to be launched at t5, and ft is updated to the beginning of the next time interval T02 (i.e., t6 instead of t_end), as shown in the right side of Fig. 6. In brief, in the second case, the scheduler updates only ft.

Finally, it is also important to note that ct should not overrun ft in any case. In other words, we require a safety margin between ft and ct. For this purpose, the run-time scheduler introduces two threshold values indicating hard and soft deadlines (d_hard and d_soft). More specifically, the scheduler monitors the time difference (t_diff) between ft and ct (i.e., t_diff = ft - ct). At the same time, the scheduler compares t_diff with the threshold values, and ft is updated to a new value (ft_new) in accordance with Algorithm 1.

The hard deadline (d_hard) is introduced to force update of ft, regardless of whether any frames are available for scheduling. On the other hand, the soft deadline (d_soft) provides some flexibility to the run-time scheduler when the time difference t_diff is between the hard and soft deadlines (i.e., d_hard < t_diff < d_soft). In this case, the scheduler is allowed to schedule a launch time for a new frame if one is available. If no frames are available for scheduling, then ft is forced to update.

In Algorithm 1, there are three constant parameters: d_hard, d_soft, and t_inc. The first two parameters (d_hard and d_soft) indicate threshold values, as mentioned above. The last one (t_inc) is related to the amount of time by which ft is incremented. Let us consider a specific case when t_diff is less than d_hard (i.e., t_diff < d_hard). Assume that the gate control list in Fig. 2 is given, and that ft is in the time interval T00 (i.e., the current time interval is T00). In this example, if ct + t_inc is within the range of T00 (i.e., between t0 and t3), then ft is updated to ft_new = ct + t_inc, as shown in Fig. 7. On the other hand, if ct + t_inc is outside the range of T00, then ft is updated to ft_new = t3 (i.e., the beginning of the next time interval, T01), as shown in Fig. 8.

4. Evaluation Setup

4.1 System Configuration

In this section, we evaluate the proposed time-aware shaper with a simple system configuration as shown in Fig. 9. The main focus here is to evaluate how accurately the timing of each transmission is controlled by the time-aware shaper. We also monitor the throughput of a best-effort stream with a data quality analyzer (Anritsu MD1230B), which is con-
Fig. 7 Illustration of $t_f$ when $t_{diff} < d_{hard}$. Both $t_f$ and $ct + t_{incr}$ are in the same time interval $T_00$. In such a case, $t_f$ is updated to $ct + t_{incr}$.

Fig. 8 Illustration of $t_f$ when $t_{diff} < d_{hard}$. $t_f$ is in time interval $T_00$ while $ct + t_{incr}$ is in interval $T_01$. In this case, $t_f$ is updated to the beginning of the next time interval, $T01$.

Fig. 9 Simple block diagram of the evaluation system. The proposed time-aware shaper is implemented in Host1.

Table 1 Specification of the host machine.

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel® Core™ i7–8700K @ 3.70 GHz (TB off)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>8 GB (4 GB × 2)</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu® 16.04.3 LTS (64-bit)</td>
</tr>
<tr>
<td>Kernel</td>
<td>PREEMPT_RT 4.9.68–rt50 [21]</td>
</tr>
<tr>
<td>Driver</td>
<td>igb_avb [20]</td>
</tr>
</tbody>
</table>

Table 2 Specification of the test streams.

<table>
<thead>
<tr>
<th>Stream Type</th>
<th>Frame Size</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-sensitive</td>
<td>64 bytes</td>
<td>125 µs</td>
</tr>
<tr>
<td>Best-effort</td>
<td>1522 bytes</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Fig. 10 Block diagram that indicates queue assignment and how latency is measured for each frame. Four timestamps ($t_{talker}$, $t_{launch}$, $t_{receive}$, and $t_{listener}$) are used to calculate latency. $t_{network}$ is defined as the difference between $t_{receive}$ and $t_{launch}$. Similarly, $l_h$ is defined as the difference between $t_{receive}$ and $t_{talker}$. Lastly, $l_{total}$ is defined as the difference between $t_{listener}$ and $t_{talker}$.

4.2 Evaluation Scenarios

One of the key requirements of TSN is to support both time-sensitive and best-effort traffic over a single standard Ethernet® network. In particular, the time-aware shaper of 802.1Qbv can be used for handling deterministic (or scheduled) message delivery for time-sensitive applications together with best-effort traffic. For ease of evaluation, we consider two types of test streams as shown in Table 2 in our test scenario. The size of each frame for the time-sensitive streams is set to 64 bytes, and timer interrupts of 125 µs are used to trigger the transmission. On the other hand, the size of each frame for the best-effort stream is set to 1522 bytes, and each frame is transmitted whenever possible to achieve high throughput.

Queue Assignment: In this evaluation, our main interest is the timing performance (i.e., accuracy) of the time-sensitive streams when interference caused by a best-effort stream exists. In order to evaluate the proposed time-aware shaper, we consider a situation in which a talker application transmits seven time-sensitive streams (from $S_7$ down to $S_1$), and another application transmits a best-effort stream ($S_0$). For illustration, we refer to the left side of Fig. 10. In the figure, Talker1 transmits time-sensitive streams through queues $Q_7$, . . . , $Q_1$. At the same time, Talker2 transmits a best-effort stream through queue $Q_0$ as shown in Fig. 10. It should be also noted that stream $S_i$ is assigned to queue $Q_i$ for $i = 0, \ldots, 7$, and a gPPT daemon is assigned to queue $Q_0$ for time synchronization.

Gate Control List: A gate control list used in the evaluation is shown in Fig. 11. The gate control list is configured with eight time intervals from $T00$ to $T07$. As shown in Fig. 11, an exclusive time interval is assigned for each time-sensitive stream in order to achieve strict temporal isolation. In par-

Connected to two host machines (Host1 and Host2 in Fig. 9) via gigabit Ethernet® cables. The specification of each host is given in Table 1. Moreover, a general-purpose oscilloscope is used to evaluate the accuracy of the time synchronization by analyzing pulse signals from the I210 NIC. It is also noted that the constant parameters used in Algorithm 1 are set as follows: $d_{hard} = 25$ µs, $d_{soft} = 100$ µs, and $t_{incr} = 100$ µs.
In an ideal (i.e., zero jitter) case, each frame of the time-sensitive stream is expected to arrive with the interval of 125 $\mu$s for each time-sensitive stream. In this evaluation, lower jitter means higher timing accuracy. We therefore compare the jitter of the time interval for Case Studies 1 and 2.

Moreover, in Sects. 5.3 and 5.4, we evaluate the frame latency for each time-sensitive stream using timestamps, as shown in Fig. 10. In particular, we calculate the network latency ($t_{\text{network}}$) and talker-side latency ($l_t$) for each frame of the time-sensitive streams. Our focus here is to compare the variation of each latency for Case Studies 1 and 2 (we are particularly interested in the peak-to-peak jitter of $t_{\text{network}}$ and $l_t$).

5.1 Time Synchronization Accuracy

In order to synchronize the clock inside the I210 NIC, the gPTP daemon [20] is activated in both host machines. In the experiments discussed in the following subsections, Hosts 1 and 2 are configured as master and slave, respectively, for the synchronization. It should be also mentioned that the gPTP daemon can provide a pulse signal from a specific pin on the I210 NIC. For each experiment, the I210 NIC is set to generate 1 million pulses per second, and the accuracy of the time synchronization is evaluated by the oscilloscope comparing the pulse signals.

Screenshots of the oscilloscope are shown in Figs. 12(a) and 12(b) for Case Studies 1 and 2, respectively. The upper side of each figure represents pulses of the master clock (i.e., Host1), whereas the lower side of each figure represents pulses of the slave clock (i.e., Host2). An edge trigger is set to capture the rising edge of each pulse generated by the master clock. Moreover, the oscilloscope is set to accumulation mode, drawing the rising edge of each pulse without removing any of the previous pulses. This helps in visualizing the accuracy of the time synchronization. Figs. 12(a) and 12(b) suggest that the peak-to-peak jitter of the clock synchronization is approximately 30 ns for both Case Studies 1 and 2. That is, the system configured for the evaluation achieves a highly accurate synchronization, on the order of less than 0.1 $\mu$s. This aspect also indicates the accuracy of the timestamps used in the following subsections.

5.2 Frame Intervals of Time-Sensitive Streams

In this subsection, we evaluate time intervals of consecutive frames for each time-sensitive stream (i.e., $S_7, \ldots, S_1$) for Case Studies 1 and 2. In order to obtain accurate time at the receiver side, we activate the time-stamping feature of the I210 NIC in Host2. With this feature enabled, Host2 records the time in a punctual manner upon the arrival of each frame of the time-sensitive streams. Using the arrival time of each frame, the frame interval of each stream is evaluated for Case Studies 1 and 2, respectively.

In an ideal (i.e., zero jitter) case, each frame of the time-sensitive stream is expected to arrive with the interval of 125 $\mu$s. For the evaluation, 1 million frames are captured from each time-sensitive stream, and frame intervals...
are calculated. At the same time, throughput of the best-effort stream $S_0$ is also monitored by using the data quality analyzer between Hosts 1 and 2. Given the gate control list of Fig. 11, the maximum theoretical throughput of stream $S_0$ is 876 Mbps.

The histograms of the frame interval of streams $S_7$ and $S_1$ are given in Figs. 13 and 14, respectively. (Due to space limitations, two typical figures are shown in the paper. We have obtained similar results for streams $S_6, \ldots, S_2$.) It should also be noted that the best-effort stream $S_0$ achieves the expected throughput (approximately 876 Mbps) when the time-sensitive streams are captured. Finally, the results obtained from each experiment are summarized in Tables 3 and 4. All values are given in microseconds.

Figs. 13(a) and 14(a) indicate that the baseline (i.e., launch-time disabled) implementation does not provide an accurate timing for transmission. In fact, the peak-to-peak values of the frame interval are in the range of 20.784 µs to 22.480 µs, as shown in Table 3. On the contrary, Figs. 13(b) and 14(b) indicate that the proposed (i.e., launch-time enabled) implementation provides almost precise timing control for transmission. In particular, the peak-to-peak values of the frame interval are in the range of 0.081 µs to 0.089 µs, as shown in Table 4. Results show that the proposed implementation can significantly reduce the peak-to-peak jitter of the frame interval compared with the baseline case, regardless of the interfering best-effort traffic (i.e., high-bandwidth stream $S_0$).

5.3 Network Latency

In this subsection, we evaluate network latency for each time-sensitive stream for Case Studies 1 and 2. At the sender side, the run-time scheduler embeds a timestamp indicating launch time ($t_{launch}$) into the payload field of each frame for streams $S_7, \ldots, S_1$. Subsequent to transmission, at the receiver side, the NIC of Host2 adds another timestamp ($t_{receive}$) to the payload field upon receipt of each frame. The network latency ($l_{network}$) is defined as the difference between $t_{receive}$ and $t_{launch}$ (i.e., $l_{network} = t_{receive} - t_{launch}$), as shown in Fig. 10. Note that each frame has its own $t_{receive}$ and $t_{launch}$ timestamps, and $l_{network}$ is calculated for each frame based on these timestamps. The main focus here is to evaluate the variation of network delays (i.e., jitter) from Host1 to Host2.

The histograms of the network latency of streams $S_7$ and $S_1$ are given in Figs. 15 and 16, respectively. In addition, the results obtained from each experiment are summarized in Tables 5 and 6. Figs. 15(a) and 16(a) indicate that the network latency is unstable when the launch-time feature is disabled. In fact, the peak-to-peak values of the network latency are in the range of 12.238 µs to 14.087 µs, as shown in Table 5.

By contrast, Figs. 15(b) and 16(b) indicate that the network latency ($l_{network}$) is highly stable when the launch-time feature is enabled. In particular, the peak-to-peak values of the network latency are in the range of 0.077 µs to 0.080 µs, as shown in Table 6. Table 6 also indicates that even the worst-case latency (max) is very stable, being less than merely 1.4 µs. These results show that the proposed implementation can achieve a deterministic network latency for time-sensitive streams.

5.4 Talker-Side Latency

In this subsection, we evaluate talker-side latency for each time-sensitive stream for Case Studies 1 and 2. At the sender side, the talker application embeds a timestamp ($t_{talker}$) into the payload field of each frame for streams $S_7, \ldots, S_1$. Subsequently, the frame is stored in a transmission queue (i.e., one of the queues $Q_i$ for $i = 1, \ldots, 7$ implemented in the host memory shown in Fig. 3). After the transmission, at the receiver side, the NIC of Host2 adds another timestamp ($t_{receive}$) to the payload field upon receipt of a frame. The talker-side latency ($l_{talker}$) is defined as the difference between $t_{receive}$ and $t_{talker}$ (i.e., $l_{talker} = t_{receive} - t_{talker}$), as shown in Fig. 10. In this experiment, the Talker1 inserts each frame
into queue $Q_7$, $\ldots$, $Q_1$ at least one time interval (i.e., 125 $\mu$s) prior to actual transmission of the frame. That is, we expect the talker-side latency to be in the range of 125 $\mu$s to 250 $\mu$s (i.e., $125 \mu$s < $l_{ts}$ < 250 $\mu$s). It should also be mentioned that the main focus here is to evaluate the variation of the latency (i.e., jitter) from the talker application to Host$_2$.

The histograms of the talker-side latency of streams $S_7$ and $S_1$ are given in Figs. 17 and 18, respectively. In addition, the results obtained from each experiment are summarized in Tables 7 and 8. When the launch-time feature is disabled, the peak-to-peak values of the talker-side latency are in the range of 41.293 $\mu$s to 48.512 $\mu$s, as shown in Table 7. On the other hand, if we enable the launch time, the peak-to-peak values are reduced to being in the range of 23.797 $\mu$s to 24.297 $\mu$s, as shown in Table 8.

<table>
<thead>
<tr>
<th>Stream</th>
<th>Min</th>
<th>Max</th>
<th>pk-pk</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_7$</td>
<td>115.728</td>
<td>137.984</td>
<td>22.256</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_6$</td>
<td>115.728</td>
<td>138.208</td>
<td>22.480</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_5$</td>
<td>115.728</td>
<td>137.056</td>
<td>21.328</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_4$</td>
<td>115.728</td>
<td>137.488</td>
<td>21.760</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_3$</td>
<td>115.728</td>
<td>137.152</td>
<td>21.424</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_2$</td>
<td>115.728</td>
<td>136.512</td>
<td>20.784</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_1$</td>
<td>115.728</td>
<td>136.512</td>
<td>20.784</td>
<td>125.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stream</th>
<th>Min</th>
<th>Max</th>
<th>pk-pk</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_7$</td>
<td>124.968</td>
<td>125.049</td>
<td>0.081</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_6$</td>
<td>124.960</td>
<td>125.049</td>
<td>0.089</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_5$</td>
<td>124.968</td>
<td>125.049</td>
<td>0.081</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_4$</td>
<td>124.968</td>
<td>125.049</td>
<td>0.081</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_3$</td>
<td>124.968</td>
<td>125.049</td>
<td>0.081</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_2$</td>
<td>124.968</td>
<td>125.049</td>
<td>0.081</td>
<td>125.000</td>
</tr>
<tr>
<td>$S_1$</td>
<td>124.961</td>
<td>125.049</td>
<td>0.088</td>
<td>125.000</td>
</tr>
</tbody>
</table>

6. Comparison of the Time-Aware Shaper and the Standard Linux® Network Stack

Our experiments of Sect. 5 show that the proposed time-aware shaper can provide precise timing control for transmission of the time-sensitive streams. These experiments, however, do not show the impact for the overall performance especially from end-users’ viewpoint. To address this issue, in this section, we investigate the effect of the proposed approach on the total latency between application-layers. In particular, we compare the performance of the proposed time-aware shaper and a queueing mechanism provided by the standard Linux® network stack.

For comparison, we implement a strict priority algo-
Algorithm based on the traffic control mechanism (i.e., queueing discipline or qdisc [22]) of the Linux® kernel. To be more specific, we define eight classes of different priority, by using the PRIO qdisc [22]. In addition, the talker application is modified to send frames through raw sockets, each of which is mapped to different class of priority (i.e., P7, ..., P0). Note that P7 represents the highest priority, and P0 represents the lowest priority. For a fair comparison, the time-sensitive streams (i.e., S7, ..., S1) are assigned to high priority classes. Specifically, stream Si is assigned to Pi for i = 7, ..., 1. Moreover, the best-effort stream S0 and the gPTP daemon are assigned to the lowest priority class (i.e., P0).

In order to clarify the importance of the proposed method in the end-to-end data delivery, we conduct experiments for the time-aware shaper (with launch-time feature enabled) and the qdisc-based implementation as described below. In this evaluation, total delay between the talker (i.e., sender) and listener (i.e., receiver) applications is calculated using timestamps, as shown in Fig. 10. At the sender side, the talker application of Host1 embeds a timestamp (t_falker) in the same way as described in Sect. 5.4. After the transmission, each listener application of Host2 records another timestamp (t_listener) upon receipt of a frame from the socket interface. It should be noted that Listeneri receives stream Si for i = 1, ..., 7, and total latency is calculated for each frame of the time-sensitive stream. The total latency (l_total) is defined as the difference between t_listener and t_falker (i.e., l_total = t_listener − t_falker), as shown in Fig. 10. In other words, l_total indicates the end-to-end delay between application-layers of Host1 and Host2. Therefore, l_total is especially important from the perspective of end-user applications.

Typical results of total latency (l_total) and talker-side latency (l_tot) of streams S7 and S1 are given in Figs. 19 and 20, respectively. In addition, the results of the total latency are summarized in Tables 9 and 10. As shown in Figs. 19 and 20, we obtained similar shapes of histograms for streams S7 and S1. When we compare Figs. 19(a) and 19(b), the mean values of l_total are 266.87 µs and 581.124 µs for the

| Table 5 | Network latency (launch-time disabled). |
|---|---|---|---|---|
| Stream | Min | Max | pk-pk | Mean |
| S7 | 5.244 | 19.331 | 14.087 | 6.165 |
| S6 | 5.201 | 19.011 | 13.810 | 6.029 |
| S5 | 5.659 | 18.683 | 13.024 | 6.407 |
| S4 | 5.548 | 18.762 | 13.214 | 6.608 |
| S3 | 5.498 | 18.434 | 12.936 | 6.444 |
| S2 | 5.703 | 18.154 | 12.451 | 6.555 |
| S1 | 5.588 | 17.826 | 12.238 | 6.677 |

| Table 6 | Network latency (launch-time enabled). |
|---|---|---|---|
| Stream | Min | Max | pk-pk | Mean |
| S7 | 1.303 | 1.382 | 0.079 | 1.338 |
| S6 | 1.303 | 1.381 | 0.078 | 1.338 |
| S5 | 1.303 | 1.381 | 0.078 | 1.338 |
| S4 | 1.303 | 1.382 | 0.079 | 1.338 |
| S3 | 1.303 | 1.383 | 0.080 | 1.338 |
| S2 | 1.304 | 1.381 | 0.077 | 1.338 |
| S1 | 1.303 | 1.382 | 0.079 | 1.338 |
time-aware shaper and the PRIO qdisc, respectively. Results show that the proposed approach can reduce the average latency of \( l_{\text{total}} \) by 54 percent compared to the qdisc-based implementation.

It should also be noted that when we repeat the same experiment many times, Host\(_2\) occasionally imposes some additional latency on a few frames (about six or seven frames out of the total 1 million frames). In particular, the delay between \( t_{\text{listener}} \) and \( t_{\text{receive}} \) is increased, and in such cases, we observed the worst-case total latency of 1.1 and 1.6 milliseconds for the time-aware shaper and the PRIO qdisc, respectively. In either case, the operating system of Host\(_2\) causes additional latency because there is no real-time guarantee on task scheduling and/or interrupt handling. In order to reduce the jitter of the total latency, any interference of the operating system should be minimized as possible. In addition, listener applications can be modified to poll the NIC for new frames instead of waiting for an interrupt request. For example, it is a good idea to utilize a kernel-bypass architecture, which is a topic for future research.

Finally, it is also important to note that the proposed approach has a significant impact on the talker-side latency (\( l_{\text{ts}} \)) compared to the qdisc-based implementation. When Figs. 19(c) and 19(d) are compared, the mean values of \( l_{\text{ts}} \) are 179.3 \( \mu \)s and 477.758 \( \mu \)s for the time-aware shaper and the PRIO qdisc, respectively. This means that the proposed approach can reduce the average latency of \( l_{\text{ts}} \) by 62 percent compared to the qdisc-based implementation. Furthermore, the peak-to-peak jitter of \( l_{\text{ts}} \) is reduced from 487.317 \( \mu \)s to 27.055 \( \mu \)s. This indicates significant reduction (about 94 percent) of the peak-to-peak jitter of \( l_{\text{ts}} \). These results show that the proposed time-aware shaper achieves highly deterministic performance (i.e., low latency and extremely low jitter) for \( l_{\text{ts}} \) compared to the qdisc-based implementation.

### 7. Related Work

The time-aware shaper compliant with IEEE 802.1Qbv requires a predefined schedule (i.e., gate control list) to achieve temporal isolation of time-sensitive streams. There are...
several publications regarding synthesis of schedules for IEEE 802.1Qbv-compliant time-sensitive networks. Craciun-
as et al. presented a method for computing static schedules for 802.1Qbv-capable network devices using satisfiability
modulo theory or optimization modulo theory solvers [23]. Dürr and Nayak presented a Tabu search algorithm for ef-
ficient computing of schedules and a schedule compression technique to reduce the number of guard bands in a sched-
ule [24]. Raagaard et al. proposed a heuristic algorithm to determine schedules at runtime for supporting dynamic re-
configuration of time-sensitive networks [25]. Oliver et al. showed how synthesis of schedules can be formalized as a
system of constraints expressed via a first-order theory of arrays [26]. Gavrilut and Pop proposed a method for syn-
thesizing a schedule based on a greedy randomized adaptive
search procedure, a metaheuristic algorithm, in which both
time-triggered and AVB traffic are schedulable [3].

Some researchers both addressed the scheduling prob-
lem and solved the routing problem at the same time. For
example, Atallah et al. proposed a fault-resilient joint topol-
yogy, routing and scheduling synthesis (JTRSS) algorithm
for IEEE 802.1Qbv-compliant time-sensitive networks [27].
The JTRSS algorithm generates a fault-resilient topology
that guarantees feasible routing and scheduling for time-
triggered traffic. Other publications, such as [5] and [28]–
[30], have addressed the synthesis of schedules and optimize
the routing of time-sensitive flows.

Latency is an important metric for time-sensitive ap-
plications. For this reason, latency characteristics of IEEE
802.1Qbv-compliant time-sensitive networks are examined

![Comparison of the histograms of total latency and talker-side latency of stream S7.](image)

![Comparison of the histograms of total latency and talker-side latency of stream S6.](image)

![Comparison of the histograms of total latency and talker-side latency of stream S5.](image)

![Comparison of the histograms of total latency and talker-side latency of stream S4.](image)

![Comparison of the histograms of total latency and talker-side latency of stream S3.](image)

![Comparison of the histograms of total latency and talker-side latency of stream S2.](image)

![Comparison of the histograms of total latency and talker-side latency of stream S1.](image)

Table 9: Total latency (time-aware shaper).

<table>
<thead>
<tr>
<th>Stream</th>
<th>Min</th>
<th>Max</th>
<th>pk-pk</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>S7</td>
<td>175.437</td>
<td>401.973</td>
<td>226.536</td>
<td>266.870</td>
</tr>
<tr>
<td>S6</td>
<td>179.902</td>
<td>406.451</td>
<td>226.549</td>
<td>271.388</td>
</tr>
<tr>
<td>S5</td>
<td>184.677</td>
<td>410.723</td>
<td>226.046</td>
<td>275.841</td>
</tr>
<tr>
<td>S4</td>
<td>189.100</td>
<td>414.804</td>
<td>225.704</td>
<td>280.398</td>
</tr>
<tr>
<td>S3</td>
<td>193.645</td>
<td>419.133</td>
<td>225.488</td>
<td>286.685</td>
</tr>
<tr>
<td>S2</td>
<td>198.391</td>
<td>426.580</td>
<td>228.189</td>
<td>293.142</td>
</tr>
<tr>
<td>S1</td>
<td>203.534</td>
<td>443.241</td>
<td>239.707</td>
<td>300.462</td>
</tr>
</tbody>
</table>

![Table 10: Total latency (strict priority).](image)
in several publications. For instance, Thangamuthu et al. compared three traffic-shaping mechanisms (burst-limiting, time-aware, and peristaltic shapers), and analyzed their worst-case end-to-end latencies [31]. Similarly, Thiele et al. presented a formal timing analysis to derive worst-case latency bounds for time-aware and peristaltic shapers [32]. In contrast to [31], they consider all blocking effects, especially those of the same priority traffic streams. Alternatively, Zhao et al. propose a method to analyze worst-case latency for IEEE 802.1Qbv-compliant time sensitive networks using network calculus [33]. Maxim and Song focus on a local delay analysis of AVB frames under hierarchical scheduling of credit-based shaping and time-aware shaping on TSN switches [34]. Finally, Hisano et al. focus on the latency issue of lower-priority streams in time-sensitive networks. They proposed a gate-shrunk time-aware shaper (GS-TAS) to decrease the latency and increase available bandwidth for lower-priority streams [35].

In order to minimize end-to-end latency, including for application layers, software tasks of end nodes should be scheduled in tight relation to the underlying time-triggered network schedule. Some researchers have addressed this issue in the context of time-triggered Ethernet® (TTEthernet) networks. For example, Zhang et al. addressed the problem of application-level (i.e., both task- and network-level) schedule synthesis and optimization for switched time-triggered networks using a mixed-integer programming multi-objective optimization formulation [36]. Similarly, Craciunas and Oliver discussed simultaneous co-generation of static network and task schedules for distributed systems consisting of preemptive time-triggered tasks, which communicate over switched multi-speed time-triggered networks [37]. In contrast to [36], Craciunas and Oliver’s approach allows preemption, which increases the solution space on the application level.

Simulation is a versatile tool for analyzing feasibility and/or applicability of time-sensitive networks. Some researchers have evaluated different use cases for IEEE 802.1Qbv-compliant time-sensitive networks. For example, Nsai et al. discussed the use of TSN (IEEE 802.1AS-Rev [18] and IEEE 802.1Qbv) in automation networks in the presence of legacy devices with respect to real-world conditions [2]. Hares et al. compared the performance of traditional priority-based approaches and a time-aware shaper in an Ethernet® fronthaul network [38]. Migge et al. focused on credit-based shapers, time-aware shapers, and the use of static priorities, and presented a case study of an automotive Ethernet network [4]. Ko et al. investigated the optimal bandwidth allocation ratio for scheduled traffic by adjusting the size of the maximum transmission unit [39]. Other works, such as [40]–[42], have presented simulation models for time-triggered traffic in time-sensitive networks.

When it comes to implementation of time-triggered communication, Steinhammer and Ademaj presented a design for a TTEthernet communication controller optimized for implementation in hardware [43]. Their implementation relies on a custom-built hardware platform. In another publication, Rumpf et al. presented a low-footprint microcontroller-based communication architecture supporting both time-triggered messages and credit-based shaping [6]. In particular, they proposed a time-aware credit-based shaper, which is a slightly modified version of the original credit-based shaper of AVB (i.e., IEEE 802.1Qaw [14]). Groß et al. proposed a scalable hardware/software (HW/SW) co-design approach for real-time Ethernet® controllers based on partitioning into communication and application components [44]. Their implementation is based on an FPGA, and supports (i) time-triggered transmission, (ii) packet reception and timestamping, and (iii) clock synchronization. Finally, Farzaneh and Knoll presented an experimental setup for benchmarking time-triggered periodic frames on the basis of the IEEE 802.1Qbv standard [45]. They were motivated by typical mixed-critical automotive applications and used industrial Ethernet® switches supporting the basic functions of 802.1Qbv in their evaluation; however, their main focus was not implementation of a time-aware shaper for IEEE 802.1Qbv. To the best of our knowledge, our work is the first to present the design, implementation, and evaluation of a time-aware shaper for IEEE 802.1Qbv by combining a software-based run-time scheduler with a time-based transmission scheme (i.e., the launch-time feature of an Intel® NIC [11]).

8. Conclusions

In this paper, we described the implementation of a time-aware shaper for IEEE 802.1Qbv [10] using a time-based transmission scheme for the NIC. The proposed time-aware shaper is specifically designed for time-sensitive networks, which require deterministic transmissions for real-time applications. We conducted evaluations to compare the timing accuracy of frame transmissions with and without the launch-time feature of the NIC. Results show that the proposed implementation can significantly (to less than 0.1 µs) reduce peak-to-peak jitter of the frame interval in comparison with the baseline. Moreover, the proposed approach can achieve a deterministic network latency (of less than 1.4 µs) for time-sensitive streams, regardless of the interference of another best-effort stream (876 Mbps). At the end of our evaluation, we also compared the performance of the proposed approach with the PRIO qdisc. Results indicate that the proposed implementation achieves outstanding characteristics in terms of both latency and jitter performance. One direction for future research is to evaluate the time-aware shaper in a multi-hop network with TSN switches. Another direction is to use a RTOS for further real-time performance.

References

OGE et al.: SOFTWARE-BASED TIME-AWARE SHAPER FOR TIME-SENSITIVE NETWORKS


Yasin Oge received his B.E. degrees in computer engineering and telecommunication engineering (double major program) from Istanbul Technical University, Turkey, in 2010. He then obtained his M.E. and Ph.D. degrees from the University of Electro-Communications, Japan, in 2012 and 2016, respectively. He joined Toshiba Corporation in 2016, and has been engaged in the research and development of network systems.

Yuta Kobayashi received his B.E. and M.E. degrees in computer science from Waseda University in 2008 and 2010, respectively. Since 2010, he has been engaged in research on communication protocol processing at Toshiba Corporate R&D Center. He is a member of IEICE.

Takahiro Yamaura received his B.E. and Master of Information Science and Technology degrees from Hokkaido University in 2004 and 2006, respectively. In 2006, he joined Toshiba Corporation, where he has been engaged in research and development on communication protocol processing. He received a Young Researcher’s Award from IEICE in 2012. He is a senior member of IEICE and a member of IEEE.

Tomonori Maegawa received his B.E. and M.S. degrees from the University of Tokyo in 1999 and 2004, respectively. In 2004, he joined Toshiba Corporate R&D Center. His research interests include computer network systems for homes, buildings, power grids, and social infrastructures. He is a member of IEICE and IEEJ.