Multiple Cyclic Queuing and Forwarding

new-finn-multiple-CQF-0921-v01
Norman Finn
Huawei Technologies Co. Ltd
September 27, 2021

Abstract

Variants of Cyclic Queuing and Forwarding (CQF, IEEE Std 8021Q-2018 Annex T) are presented. CQF requires that all nodes run at the same frequency, but does not require that all output ports' cycles be in phase. Using three buffers (or more) allows for long links, and supports short/long path recombination. Running multiple instances of CQF on one port at different cycle rates can give good latency and bandwidth utilization for a mix of Streams with very different bandwidth requirements. These improvements require little or no alteration of IEEE 802.1Q, and like the original 2-buffer CQF, none require per-hop per-Stream dynamic state. These ideas are presented for possible inclusion into an amendment to IEEE Std 802.Q. The determinism of CQF is demonstrated, and a set of parameters is suggested for network management purposes.

1 Introduction

NOTE: This document is substantially the same as df-finn-multiple-CQF-0919-v02, the principle difference being the addition of section 1.4, Multi-CQF vs. CQF, and section 4, Deterministic behavior of Multi-CQF.

The remainder of section 1 defines the domain of interest of this paper. Section 2 provides a detailed timing model for Cyclic Queuing and Forwarding (CQF, IEEE Std 802.1Q-2018 Annex T). It shows how two, three, or more buffers can be used to manage the allocable bandwidth and per-hop delay, and why the systems' CQF cycles do not have to operate in phase. Section 3 shows how multiple instances of CQF can be run on the same output port, with different cycle times, in order to efficiently serve Streams with a wide range of bandwidth and latency requirements. Section 4 analyzes the determinism of CQF and suggests a set of control parameters, and section 0 suggests a number of further augmentations that can be made to CQF to improve bandwidth utilization and worst-case latency.

This paper assumes the reader is reasonably familiar with CQF as described in IEEE Std 802.1Q Annex T. The frame timestamps used in this paper are described in IEEE Std 802.3-2018 clause 90. Preemption, or interspersed express traffic, is described in IEEE Std 802.3-2018 clause 99 and IEEE Std 802.1Q-2018 clause 6.7.2.

1.1 Deterministic Quality of Service

TSN (and IETF DetNet) supply a Quality of Service (QoS) to a critical data flow, or "Stream". This QoS is:

a. An absolute upper bound on the end-to-end latency to frames belonging to the Stream. (Bounded Latency); and
b. A guarantee that no frames of the Stream will be discarded due to a buffer being full when the frame arrives at an intermediate hop (Zero Congestion Loss).

This QoS, which we will call the Deterministic QoS, is made possible by a promise, made by the source of a Stream, to not exceed a contracted bandwidth and maximum frame size. This guarantee allows the network to run a resource reservation procedure that dedicates resources to a particular Stream (or sometimes, to a class of similar Streams) at every hop through the network, before the first frame of the Stream can be transmitted.

1.2 Continuous Streams

We can divide the Streams that can make use of the Deterministic QoS into two classes:

- Continuous Streams can be usefully characterized by a maximum frame size, and a maximum bandwidth.
- Scheduled Streams transmit on a regular, repeating schedule.

Note that these two categories do not encompass all possible data flows. Bursty, irregular flows are not Streams, in the sense that it is difficult, in the presence of multiple of these flows, to guarantee Deterministic QoS except by overprovisioning and an extensive analysis of worst-case inter-Stream interference scenarios.

Scheduled Streams can be handled by using IEEE Std 802.1Qbv (now IEEE Std 802.1Q-2018 clause 8.6.8.4, Enhancements for Scheduled Traffic) to schedule frames in detail. They are of no interest to this paper. This paper Is concerned only with continuous Streams.

1.3 Store-and-forward

We will deal here only with store-and-forward systems, where whole transmission units are received, enqueued, and forwarded on another link. That is, we will not attempt to reconcile

CQF with cut-through forwarding. We will, however, consider ways in which frames can be subdivided and/or preempted to reduce the size of transmission units.[1]

1.4 Multi-CQF vs. CQF

The differences between CQF, as defined in Annex T of IEEE Std 802.1Q-2018, and Multi-CQF, as defined here, are:

a) For a given value of the cycle time T_C on a given output port, Annex T provides two buffers, each implemented as a separate class of service queue. Scheduled output gates (8.6.9 of IEEE Std 802.1Q-2018) are configured to enable the two queues to output alternately, each given time T_C to drain.

   Multi-CQF allows two or more buffers per output port per cycle time. It is not practical to dedicate one of only eight available class of service queues for each of these buffers. A suggestion has been made (see new-finn-pulsed-queuing-0821-v03) to enable one class of service queue to provide any number of CQF buffers for a single cycle time.

b) Annex T assumes that one value of T_C is sufficient for any given output port.

   Multi-CQF allows multiple values of T_C, one for each Multi-CQF class of service. The output cycles are constrained, on any given port, so that an integral, and never a fractional, number of shorter cycles are contained within any given longer cycle.

c) Annex T uses synchronized time so that every output port in a TSN network switches buffers simultaneously.

   Multi-CQF allows each bridge to perform its buffer switching at different times, subject to the above constraint b).

d) Annex T uses the same cycle time and phasing for the input gates as for the output gates. The input gates select which of the two output buffers on a given port to store a received frame.

   Multi-CQF runs the same cycle time for input and output gates, but adjusts the phase of the input gates on a port to match the phase of the frames arriving from the output gates of the bridge transmitting to that port.

2 CQF timing model

We have two nodes, A and B. Both are running an instance of Cyclic Queuing and Forwarding on each of multiple ports, more-or-less as described in IEEE Std 802.1Q-2018 Annex T. We will assume that nodes A and B are synchronized with each other in time to some accuracy that is

[1] The reader may also be familiar with the terms, “bulk streams” and “intermittent streams,” defined in 7.1.1 of IEEE Std 802.1CB-2017. These terms have meaning only in the context of choosing an algorithm for use by the 802.1CB sequence recovery function, and are not used, here.

significantly smaller than the CQF cycle time. We do not assume that the output buffers switch in synchrony; they can be out of phase.

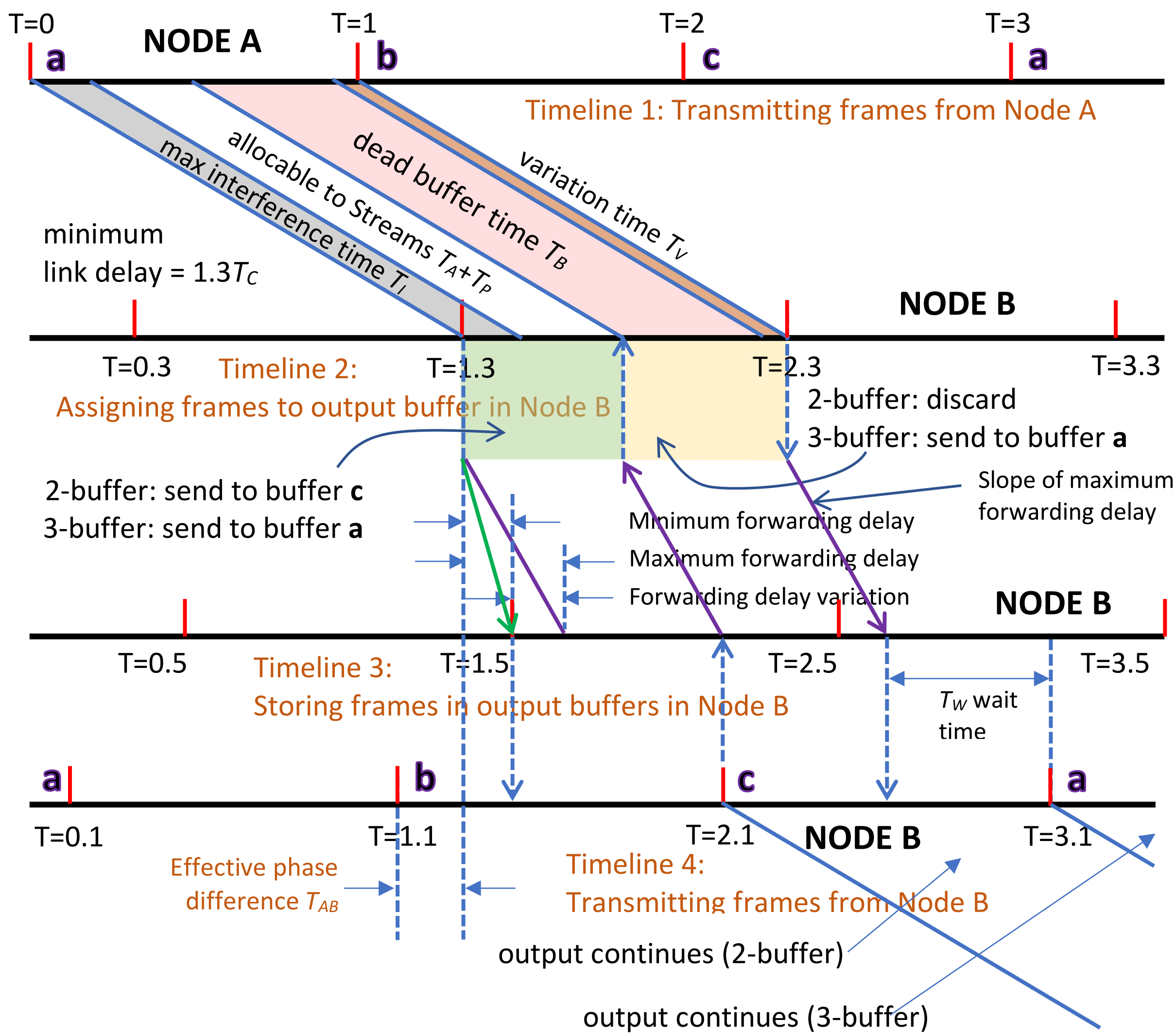
After a gate open/close event on a particular port, node A transmits all of the frames in one cyclic buffer towards receiving node B, not necessarily in a single burst. After some gap following the transmission of the last frame in the buffer, another gate open/close event is performed. At this point, it starts transmitting the frames from the next cyclic buffer. The gate open/close events in both nodes happen regularly, with the same period T_C. At the next hop, node B must be able to assign each received frame to a transmit buffer such that 1) frames that were in the same buffer in node A, and are transmitted on the same port from node B, are placed into the same buffer in node B; and 2) frames in different buffers in node A are placed in different buffers in node B.

Figure 1 shows an example of Cyclic Queuing and Forwarding. Node A and Node B are transmitting at the same frequency, but are offset by $0.1T_C$, as shown by timelines 1 and 4. In Figure 1, we use the following notation for time intervals:

T_C nominal (intended) period of the buffer-swapping cycle
T_I maximum interference from lower-priority queues, one frame or one fragment
T_V sum of the variation in output delay, link delay, clock accuracy, and timestamp accuracy
T_A the part of the cycle allocable to (reservable by) Streams
T_P worst-case time taken by additional bytes if this traffic class is preemptable
T_B end-of-cycle buffer dead time optionally imposed on node A by node B
T_W wait time during which buffer is neither receiving nor transmitting frames
T_{AB} effective phase difference between cycle start times for input from A and output from B

Following the definitions of output gates in IEEE Std 802.1Q-2018, the red ticks in timelines 1 and 4 in Figure 1 represent the earliest possible moment at which the first bit of the destination address of the first frame of the cycle can be transmitted. These ticks are driven by the synchronized clock. They are the basis for all cyclic buffer transmissions. If Enhancements for Scheduled Traffic (ETS, IEEE Std 802.1Q-2018 8.6.8.4) are used for controlling the output buffers, the ticks are the points in time when the output gate of one queue is closed, and the next queue's gate is opened. These are the points in time as programmed into the managed objects that control ETS. An implementation may need to schedule events in anticipation of the time specified in the managed objects in order to maximize throughput. Note that the preamble of an IEEE 802.3 Ethernet frame can be transmitted before gate open event.

Figure 1 Reference timelines for time-based CQF



2.1 Output timeline 1

Figure 1 shows an interference delay T_I (the gray area) between the gate event (the red ticks in Figure 1) and the transmission of the first bit of the first Stream frame's destination MAC address. The interference is from frames transmitted from lower-priority queues. It is equal to the time required for one maximum-length transmission unit over all lower-priority queues. That maximum transmission unit is either a maximum-length fragment, for preemptable lower-priority queues, or the maximum-length frame, for non-preemptable queues. The value of T_I depends upon the configuration of lower-priority queues.

It is possible that the class of service illustrated in Figure 1 is, itself, a preemptable class. In that case, a higher-priority class of service can preempt transmission of frames in this class.

Preempting a frame adds additional bytes to the resultant fragments, which must be accounted for when allocating bandwidth to a class of service. T_P represents the worst-case additional time required to transmit these extra bytes caused by preempting frames belonging to a CQF Stream. This value is always bounded. See below, section 3.2.

There can be some variation in the time from the selection of a frame for output in node A to the timestamp moment, when the first bit of the destination MAC address is transmitted (see IEEE Std 802.3-2018 clause 90). This is called output delay variation. The total time between the transmission of the first bit of the frame and the reception of that first bit at the next hop is called the link delay. Depending on the medium and the length of the link, there can be variations in link delay. The worst-case variation between the two node's clocks caused by accumulated frequency variations, asymmetrical links, etc., causes uncertainty between the transmitting and receiving nodes' clocks, and in the determination of the link delay. The inaccuracy in converting between IEEE 802.3 transmit and receive timestamps and the local clock that drives the gate open/close events also contributes to cycle accuracy. The worst-case combination of these four items, output delay variation, link delay variation, clock/frequency uncertainty, and timestamp conversion inaccuracies, is labeled, T_V.

All of the contributions to T_V are lumped together at the end of the cycle, even though contributions to T_V are made throughout the cycle.

As described below (section 2.4) the next hop can impose a buffer dead time T_B on this hop. This is a time at the end of the cycle, during which no frames can be transmitted from the cyclic buffer, so that the last frame of the cycle can be received earlier than the end of the cycle.

It is necessary, in order to know how much data can be transmitted in one cycle, that an implementation be able to transmit all of the frames in a cyclic output buffer together, at line rate, with no interference from lower-priority queues on the same output port. (Interference from higher-priority queues is described in section 3.2.) Given that is true, then the total time per cycle that can be used for transmitting Streams is:

$$T_A = T_C - T_I - T_P - T_B - T_V.$$

This T_A is a maximum, local to a particular output port on a node. It guarantees that the last frame of cycle (plus a possible preamble of the first frame of the next cycle) will be on the wire before the output gate closes. All of the components of T_A can be calculated by an implementation from its configuration and from knowledge of the implementation, except for T_B and parts of T_V. T_B is supplied by configuration, or by the node to which the output port is connected. T_V can be supplied either by the time sync implementation, by configuration, by summing the contributions of node A and node B, or by the specification of a maximum allowed value by a standard or an equipment purchaser.

Note that T_A, as defined here, includes the entire transmission time of Stream data, including one 12-byte inter-frame gap and one 8-byte preamble for every frame. The preamble of the first frame of a cycle is counted in the previous cycle due to the way in which the output gates are defined in IEEE Std 802.1Q.

2.2 Receive timeline 2

The timeline at the receiving port is timeline 2 in Figure 1. The red ticks represent the earliest possible moment that the first bit of the destination MAC address of the first frame of a cycle can be received. In terms of IEEE 802.1Qci (IEEE Std 802.1Q-2018 clause 8.6.5.1 Per-Stream Filtering and Policing), a timed input gate must open no later than this point.

On timeline 2, each frame is assigned to a buffer on an output port based on the timestamp (IEEE Std 802.3-2018 clause 90) on the frame.

A critical aspect of timeline 2 is its offset from timeline 4, the output timeline. This offset is shown as T_{AB} in Figure 1. It is clear from the figure that T_{AB} must be known in order to compute T_B and T_W. T_{AB} can be computed by 1) synchronizing the clocks of nodes A and B, and 2) measuring the link delay from node A to node B using PTP. Other methods are also possible, e.g. that described in [1-21-0056-00-ICne-input-synchronization-for-cyclic-queueing-and-forwarding].

Once T_{AB} is known, all of the timing relationships shown in Figure 1 can be computed. The phasing of the nodes' output buffer cycles certainly does affect the end-to-end latency of any stream, so that phasing must be known when the latency is computed. The end-to-end latency is no longer an integer multiple of the cycle time. It is even possible to adjust the phasing to favor certain paths through the network.

For a node B that is connected to and receiving cyclic frames from *n* other nodes, we have *n* assignment problems to solve, one for each input port on node B.

If a frame is received that straddles a cycle (first be in one cycle on timeline 2 of Figure 1, and end frame plus inter-frame gap plus a preamble time occurs in the next cycle), then either 1) some part of that frame was transmitted from node A outside the cycle window T_C, or 2) one or more of the constants, measurements, or calculations above is incorrect. Either way, the frame must be discarded, or else it can cause disruption of delivery guarantees farther along in the network.

2.3 Storing frames timeline 3

The timeline at the point where frames are stored into an output buffer is timeline 3 in Figure 1. The red ticks on timeline 3 mark the earliest point at which the first frame transmitted from a particular buffer could reach the output buffers (neglecting transmission time on the input medium). These ticks are offset from timeline 2 by the minimum forwarding delay, required to forward the frame from the input port to the output queue. The maximum forwarding delay is also shown. The forwarding delays shown in Figure 1 include the time to install the frame in the output buffer and for its presence to filter through to the point that it can be selected for output.

There are two possible buffer assignment methods shown in Figure 1: the two-buffer method, in which the frames received from node A buffer a are assigned to buffer c in node B, and the three-buffer method, where those same frames are assigned to buffer a in node B. The slope of the maximum forwarding delay allows us to compute the latest moment at which frames received from buffer **a** on node **A** can be stored into buffer **c** on node **B**. The shaded areas just below timeline 2 in Figure 1 show the time windows for buffer assignment. If two output buffers are used, then frames received from buffer **a** on node **A** can be assigned on input (timeline 2) to buffer **c** only as long as they are assured of being placed into buffer **c** before node **B** starts transmitting buffer **c**. As shown, frames from buffer **a** can be assigned to buffer **a** (three-buffer mode) during the entire length of the cycle on timeline 2. Time T_W in Figure 1 is the time during which, in three-buffer mode, buffer **c** is holding frames, neither filling nor emptying. In 3-buffer mode, the dead buffer time T_B is 0, and T_B, the allocable transmission time, encompasses both the T_B (white) and T_B (red) regions in Figure 1.

2.4 Calculation of T_B

Timeline 3 in Figure 1 shows the calculation of T_B, which applies only to two-buffer mode. The starting point of is the moment that the output cycle starts (the tick on timeline 4), backed up by the worst-case forwarding delay. This is the last moment on timeline 3 that a frame can be assigned to buffer **c** in the example in Figure 1. The end of T_B is the end of the cycle T_C, less the variation time T_V. In three-buffer mode, T_B is zero.

T_B is only known to node B. Its effect on the allocable bandwidth T_A must be taken into account when admitting new Streams. If a network uses a peer-to-peer control structure using, e.g. IEEE Std 802.1Q-2018 MSRP, then the value of T_B must be made available to the previous node A so that node A does not exceed the reduced T_A.

There are many ways to deal with this issue. Here are three:

1. The value of T_B can be propagated backwards to the previous node, either via management or via an extension of the reservation protocol.

2. A node can compute the value of T_B and decide whether to employ 2-buffer or 3-buffer mode, depending on how much bandwidth has been allocated, so far. This, of course, can change previously-computed Stream's end-to-end latency.

3. All nodes in a network can be configured with a reasonable maximum value for T_B. If a particular input/output port pair on a particular node computes a value for T_B that exceeds this maximum, then 3-buffer operation is required.

2.5 Transmitting frames timeline 4

Depending on whether two-buffer or three-buffer mode is used, one can trade off reduced total available bandwidth against per-hop delay. Timeline 4 in Figure 1 shows the two options for the choice of which output cycle in node B is used to transmit frames that were transmitted from buffer **a** in node A.

2.6 More than 3 output buffers

The discussion over Figure 1, so far, assumes that the variation in forwarding delay is small, relative to T_C. If this is not the case, node B can use more than 3 output buffers, and assign received frames to buffers whose output is scheduled far enough ahead in time to ensure that, in the worst case, they will arrive in the buffer before the buffer begins transmitting. This works only because the buffer assignment decision is made based on time-of-arrival of the frame at the input port, not the time-of-arrival of the frame at the output port.

In certain situations, e.g. when Stream is split and traverses two paths of different lengths using IEEE Std 802.1CB Frame Replication and Elimination for Reliability (FRER), it can be desirable to purposely delay a Stream's frames in order to match the total delay for the Stream along the two paths. In this case, more than 3 output buffers can be allocated, and used to impose a delay of an arbitrary number of cycle times T_C on every frame.

This author claims, without a demonstration, that it is not difficult to implement CQF so that each output port in a node, and each output port along the path of a Stream, can have a different number of buffers, whether 2, 3, or 50. Not only that, but one flow can use 3 buffers on an output port, while another flow, which needs a path-matching delay, can use 12 buffers on the same port. (Of course, this would require per-flow configuration.)

3 Multiple CQF classes of service

3.1 Multiple T_C model

With CQF as it is described in IEEE Std 802.1Q-2018 Annex T, we are limited to a single class of service (a single value of T_C) and to 2-buffer operation, only. We have already discussed 3-buffer (or more) operation. We will now discuss the simultaneous use of more than one value of T_C on the same output port.

It can be difficult to pick a single value of T_C for a network. If the chosen value is small, then only a few Streams can be accommodated on any one port; all frames for all Streams must fit into a single T_C period. If the value chosen for T_C is large, then more Streams can be accommodated, with a wide variation in allocated bandwidth, but the larger T_C increases the per-hop latency. In the ideal case, of course, every Stream would have a T_C value chosen so that exactly one frame of a Stream is transmitted on each cycle T_C.

While this is not always possible, we can apply multiple values of T_C to a single output port, as shown in Figure 2.

Figure 2 Multiple T_C *values*

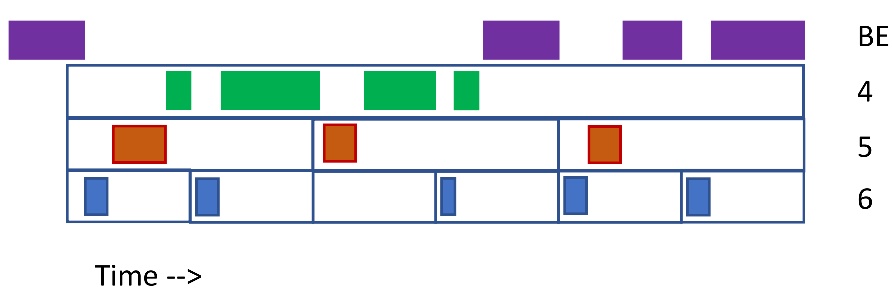
<table>
<tr><td colspan="24">h</td><td>3</td></tr>
<tr><td colspan="8">f</td><td colspan="8">g</td><td colspan="8">f</td><td>4</td></tr>
<tr><td colspan="4">d</td><td colspan="4">e</td><td colspan="4">d</td><td colspan="4">e</td><td colspan="4">d</td><td colspan="4">e</td><td>5</td></tr>
<tr><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>6</td></tr>
</table>

In Figure 2, we have a schematic timeline. We are running four values of T_C simultaneously. The fastest (call it, "T_{C6}") runs at the highest priority (6). T_{C5} is slower by a factor of 4 from T_{C6} in this example, and its buffers run at priority 5 (less important than priority 6). T_{C4} is slower by a factor of 2 from T_{C5}, and by a factor of 8 from T_{C6}. T_{C3} is 24 times slower than T_{C6}. The letters in Figure 2 label which buffer is output during the cycle. There are 9 buffers a through i (buffer i is not shown). In this example, priority 6 uses three buffers, because the timing is tight; the others use two each.

We assume here that the receiver of a frame can identify the particular CQF instance (T_C value) to which the frame belongs by inspecting the frame. A TSN bridge could use the L2 priority of the field, for example. An IP router could use the DSCP. IEEE Std 802.1CB and IEEE Std 802.1Q provide for the use of other fields in the frame, e.g. IP 5-tuple.

Since the total bandwidth of the link is not oversubscribed by Streams, each cycle, fast high-priority and slow low-priority, is guaranteed to be able to transmit all of its frames within the duration of its cycle. For example: If 50% of T_{C5} is reserved, and 30% of T_{C3} is reserved, then 80% of the total bandwidth has been reserved, leaving only 20% for other Streams, best effort traffic, and dead time. This is shown in Figure 3, where we illustrate the timing of transmission of frames from three levels of CQF and the best-effort (BE) level. Note that CQF traffic can be delayed within its window by interference from both higher priorities (e.g. the first priority 4 frame) and lower priorities (e.g. the first priority 6 frame), but that it will always get out before the window closes, assuming that the bandwidth is not oversubscribed.

Figure 3 Transmission timing



As is normal with CQF, a given Stream is allocated a fixed number of bits that it can transmit per cycle T_{Cn}. Each Stream is assigned to the highest-numbered (fastest) CQF instance such that, at the Stream's bandwidth and frame size, the Stream is fast enough to occupy space in every buffer at that level. Then, CQF will maintain one or two frames in its buffers per Stream, the best possible latency is given that Stream, and the buffer space is not wasted in unused cycles.

Of course, it is the "best possible" latency only to a certain extent. The potential mismatch between the Stream's frame rate and frame size to the available values of T_{Cn} requires some overprovisioning.

Streams are allocated to, and thus use up the bandwidth available to, each cycle separately. Any cycle can allocate up to 100% of the bandwidth of that cycle's T_A, but the percentages allocated to all of the cycles must, of course, add up to less than 100%. The total amount of buffer space required depends on the allocation of Streams to priority values. If all Streams are slow and are allocated to T_{C4} up to a total of 100%, then full-sized buffers must be used for buffers h and i. If all Streams are fast and are allocated to T_{C6}, then only three small buffers are used—buffers a, b, and c are rapidly re-used.

NOTE—There are many ways to allocate buffer space to individual frames. Running CQF at 5 levels does not increase the buffer memory requirements beyond that of 1-level CQF. Allocating bandwidth to slow cycle times uses more buffer space, of course, because frames dwell for a longer time. This is inherent in any scheme that offers comparable low-bandwidth high-delay service.

Given the ideal allocation described, each Stream is allocated one frame in each cycle of one row. It thus gets the optimal latency for its allocated bandwidth, which may be somewhat oversubscribed. If the end-to-end latency requirements of the Streams permit, a Stream can be assigned to a slower (lower-numbered) cycle. This will reduce the overprovision factor, since the overprovision factor depends on the number of frames per cycle. It also increases buffer usage, of course.

Any such overprovision can equally be thought of as an increased latency for that same Stream. That is, if that oversubscribed Stream was the only Stream, then the T_C cycle time could be shortened to exactly the point of 1 frame per cycle, with 0 overprovision, and thus give a faster latency. Overprovision = lower latency, in this case.

The maximum reserved bandwidth is supported by allocating a Stream multiple frames per cycle, as allowed by the Stream's required end-to-end latency, thus minimizing overprovision.

3.2 Preemption and interference

Frame preemption is described in IEEE Std 802.3-2018 clause 99 and IEEE Std 802.1Q-2018 clause 6.7.2. Not all of the bandwidth in a cycle T_C can be allocated. The smaller the cycle time, the greater the impact of the interference time (T_I in section 2 and Figure 1) on the allocable bandwidth.

T_I is equal to the worst-case transmit time for a single transmission from a lower-priority queue. This interference can occur only at the beginning of a cycle. Since this value must obviously be bound, it places a requirement, that must be enforced, on all lower-priority queues that they either have a maximum frame size or that frame preemption is applied to the lower-priority queues. If preemption is used, the maximum interference is the maximum fragment size (about 150 bytes, see IEEE 802.3). The interference time is shown as a gray parallelogram attached to timeline 1 in Figure 1.

The other time is the preemption time T_P, which applies only to Streams that are preemptable. This case is not typical, but is possible if a large fraction of the available bandwidth is to be assigned to one or a few high-bandwidth Streams, and lower-priority Streams use larger frames. T_P is the product of (the maximum number of highest-priority transmission windows that can open during a single window for the level being computed) * (the per-preemption penalty). Thus, in Figure 2, if priority 4 is preemptable, then there are 8 level 6 windows that can open. This means that there can be 8 preemption events during one level 4 window, so the total preemption time T_P is 8 times the preemption penalty. (It doesn't matter which specific frames are preempted; only how many such events occur.) The preemption penalty is the number of bytes added when a frame is preempted, which is 4 (CRC on preempted fragment) + 20 (inter-frame gap) + 8 (preamble for continuation fragment) = 32 bytes.

3.3 T_C computation

If the time per cycle that is allocable to Streams is T_A, then we can now state the computation for T_C, given T_A, or for T_A, given T_A, at each level in Figure 2:

$$T_C = T_A + T_P + T_I + T_B + T_V$$

The sum of all Stream's bits-per-cycle allocation must be less than or equal to T_A.

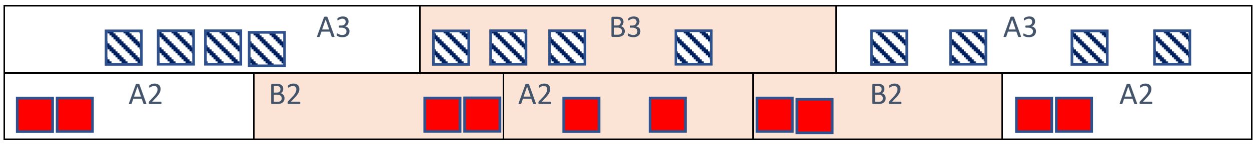
IEEE Std 802.1Q-2018 Annex T, assumes the 2-buffer scheme, and so assumes that T_B and T_V are small enough and T_C large enough to leave a useful T_A. Assuming that one's goal is the smallest possible T_C:

a. T_B can be eliminated by using the 3-buffer scheme.
b. Implementation steps can be taken to reduce T_V. This may include steps to reduce the variability of the forwarding delay, the delay between selection-for-output and first-bit-on-the-wire at the previous hop, or increased accuracy of the synchronized clock.
c. T_I can be reduced by restricting the maximum frame size of lower-priority Streams, or by enabling frame preemption.

3.4 Why integer multiples for T_C?

The ideal would for each Stream S to have its own T_{CS} that gives no overprovision. But, that winds up being equivalent to a per-Stream-shaper solution such as Asynchronous Traffic Shaping or IntServ. The reason can be seen in Figure 4.

Figure 4 Variable T_C



In Figure 4, we have allocated 40% of the link bandwidth to the solid red Stream in cycle 2, and 50% of the link bandwidth to the blue striped Stream in cycle 3. The cycles do not line up with an integral number of faster cycles in each period of slower cycle. Since we cannot predict exactly where, during a cycle, frames can be emitted, we can get the situation shown, in the shaded boxes. Buffers B2, A2, and then again, B2 emit their frames (at high priority) at the indicated times. Even though the solid red Stream takes up only 40% of each level-2 cycle, it can output 6 frames over the course of cycle B3, thus taking up 60% of the bandwidth during that period. There is, therefore, 110% of the bandwidth that must be output during the period that B3 is transmitting. B3 cannot output all of its data. Some of it must be somehow delayed, but there is no place to put that data. TSN fails.

Having an integral number of cycles at each layer fitting into one cycle at the next-slower layer ensures that the lower-priority, slower cycle, will always have sufficient time to output all of its frame, because the problem in Figure 4 is avoided. It also bounds the number of preemption events that can steal bandwidth from a given priority level.

4 Deterministic behavior of Multi-CQF

4.1 Basic requirement for determinism

Multi-CQF guarantees the Deterministic QoS by the following argument.

We assume that the Talker uses Multi-CQF. Section 4.1.1 completes the argument when this is not the case.

We consider only one value of T_C along the path of a given Stream from Talker to Listener. Sections 4.1.2 and 4.1.3 deal with exceptions to this assumption.

The contract between the Talker and the network is in terms of 1) a maximum frame size, and 2) a maximum number of bit-times on the medium per cycle time. For Ethernet, the number of bit times for a given frame is equal to (the frame size from destination MAC address through Frame Check Sequence, plus 20 bytes for preamble and inter-frame gap) times 8 bits per byte.

A number of considerations reduce the fraction of the total time T_C that can actually be used to transmit data. See section 2 for details. For example, the maximum frame size of each Stream allows us to determine the worst-case interference that a given Stream can have on higher-priority streams. All of these considerations are bounded; if an implementation cannot bound one or more of these considerations, then it cannot guarantee the Deterministic QoS in a CQF network.

In a detailed timing analysis, we will assume that the primary rule of IEEE Std 802.1Q Scheduled Transmissions is adhered to: the first bit of the preamble of a frame is never transmitted before the start of the window time (according to the local time in the transmitter) and the last bit of the interframe gap is transmitted before the end of the window.

In order to obtain Deterministic QoS for each Stream, we must ensure that no buffer is ever asked to hold more data than it can transmit during one cycle time T_C. Since the amount of data supplied by any given Stream in one cycle is set by contract, we can accomplish this as follows:

1) The Talker contract is enforced when a Talker's frames are first placed into a CQF output buffer after entry to the network. That is, the frames from a given Stream do not exceed the Talker contract in the first CQF output buffer in the network.

   As explained in dd-finn-CQF-and-shaping-0120-v01, Mick Seaman's Paternoster algorithm offers an excellent means of ensuring that this criterion is met.

2) Frames belonging to the same Stream that are in the same CQF output buffer in one bridge in the network are placed in the same CQF output buffer in all subsequent bridges along a shared path.

   Section 2, and particularly Figure 1, show the details of how this is accomplished. The key is to get the input gates synchronized with the output gates of the transmitting system, offset by the link delay. Frames received during one input cycle are always placed in the same buffer. If the input cycle is synchronized with the previous hop's output cycle, then cycle integrity is maintained. (Of course, this only works for point-to-point links.)

3) Admission control ensures that, on any given output port and cycle time T_C, the total bits times for all Streams passing through that port and T_C value does not exceed the available transmission time on that port. (This assumes that no bridge has a limitation

on available *receive* time on an input port that is smaller than the attached output port's available transmit time. The implications of such a limitation are obvious.)

4.1.1 Non-CQF Talker

As explained in dd-finn-CQF-and-shaping-0120-v01, the per-flow shaper of Mick Seaman's Paternoster algorithm offers an excellent means of ensuring that the first CQF buffer has no more than the contracted amount of data for a given Stream. It meters no more than the allowed bit-times worth of frames into a CQF buffer. This can serve both to buffer small bursts, and to enforce the contract by discarding violating frames or marking them down to best-effort.

When changing from one contract specification (e.g. leaky bucket) to a CQF specification (bits per cycle), some over-provisioning is typically required in order to ensure against congestion loss.

4.1.2 Admission control for multiple T_C values

Section 3 describes the operation of Multi-CQF with multiple T_C values operating simultaneously on one output port. Figure 3 shows an example of a sequence of transmissions. We observe that the shortest cycle times operate at the highest priority, and the longest at the lowest priority. Because different CQF priority levels may have different maximum frame sizes, and because some may enable preemption, different priority levels may have different amounts of time during one cycle that cannot be allocated to Stream transmission. Clearly, allocating time for any CQF priority level reduces the time allocable to other priority levels; there is only one physical link.

An administrator may wish to restrict allocation of CQF transmission times to leave room for transmitting non-CQF frames, either best-effort traffic or other, lower-priority TSN traffic.

For a new Stream to be admitted, it must be true that the available transmission times over all of the CQF levels on all of the output ports through which the Stream travels have not been exhausted. At any given CQF priority level *x*, one can add the bits allocated to all Streams in one cycle at CQF priority level *x*, plus the sum over all more-important CQF priority levels *y* (faster cycles), of the product of the number of bits per cycle allocated at that level times the number of cycles at that level contained within one cycle at level *x*. At every level, the total must not exceed the maximum number of allocable bits at that level.

(This calculation is simpler if, at every CQF priority level, there is the same percentage of dead time and slop for inaccuracies, but this is not necessarily the case.)

4.1.3 Changing T_C values along the path of Stream

If a Stream enters a bridge using a cycle time T_C, and is being transmitted on an output port with cycle time $n*T_C$, then n successive input cycles can be deposited in the same output buffer with no problem, as long as the larger cycle time's dead time requirements are met. (This is not a trivial exception, as the larger cycle's dead time occurs at the end of the large cycle, and thus may take up much or even all of one small cycle.) Equivalently, the input port can be configured with the slower cycle time to match the output port in the same system. Of course, when making the reservation for that Stream, the adjustment of its contract must be made; it is allocated *n* times the number of bits in the slower cycle than in the faster cycle.

In all other cases, when a Stream changes cycle times, the Stream must pass through a conditioning step, such as a Paternoster shaper (see section 4.1.1), to ensure that the Stream never exceeds its contract in the new cycle time.

4.2 Computing the actual end-to-end latency for Multi-CQF

After adjusting to get the receiving window aligned with the previous-hop transmitting window, a bridge knows the "effective phase difference TAB" described in section 2. Referring to Figure 1, this allows the bridge to compute the difference, in time, between the start of an input window for the Stream, and the start of the output window in which a frame received in that input window will be transmitted. This is the dwell time for the frame in this bridge. Adding this to the one-way link delay gives the per-hop delay for frames in the Stream. At egress from the CQF network, there is a margin of one cycle time less one frame transmission time for delivery of the frame, as the frame can be transmitted at any point during the cycle, but must both start and finish its transmission within the cycle. The delay at ingress is somewhat more complicated to measure, as it depends upon the method used by the Talker and the ingress bridge to shape its transmissions.

If we look again at Figure 1, we can see that the difference between using two and three buffers for a given input-output port pair is really a matter of rounding up the link delay to an integral number of cycle times. If the sum of link delay and phase delay between output cycles is negligible, or happens to be very nearly an integer multiple of the cycle time, then the yellow "discard" area is small, and two buffers can be used. If sum is larger, then one necessarily chooses between a smaller allocation (large discard area) and increased delay.

4.3 Per-Stream and per-port static and dynamic state

Paternoster requires per-stream dynamic state, as each frame's contribution to its Stream's bit-time allocation must be counted as the frame is deposited into a cycle buffer.

In the general case, the number of buffers required (usually 2 or 3) depends on the relative phase of the input and the output cycle start times. But different input ports generally will have different phases. Thus, the number of buffers used by any given output port will vary with the input port; an output port can have three buffers, for example, but for some input ports, there are never frames from that port in more than two buffers.

For the present time, we will assume the following method for receiving a frame and assigning it to a buffer. It is important to stress that there are many ways to accomplish the same task.

Let B_o be the number of physical output buffers on port o. We compute N, the least common multiple over all B_o in the system. Each input port i assigns each received frame a buffer selector S, which is an integer in the range 0 through N—1, and which increments (modulo N) each input cycle. Thus, frames transmitted from the same buffer are assigned the same S value at the receiving end of the link.

At the output port o, each of the B_o buffers is identified by a buffer number in the range 0 through B_o—1. A variable X_o indicates which buffer is currently transmitting. X_o increments once modulo B_o each output cycle.

When a frame arrives at an output port, it is assigned to a buffer b using the formula:

$$b = (S + P_{io}) \bmod B_n$$

Where P_{io} is the cycle phase offset from input port i to output port o. See 4.5.4 for the determination of P_{io}. Note that in the extreme case of all output ports using two buffers, all synchronized, and all input cycles in phase with the output cycles, the table P_{io} reduces to a single value, 0 or 1, and we have CQF from Annex T.

It is desirable in some cases to deliberately use more buffers than are required for insurance against congestion loss in order to match the end-to-end delay of a Stream across different paths through the network. If such delay matching is performed per-Stream, instead of per-input port, then per-stream P_{io} values are required for buffer selection.

P_{io} is not dynamic, though its values may change when the relative phasing between an input port cycle and the transmitter feeding it change suddenly. Such a change will always disrupt the CQF service guarantees.

4.4 Implementation requirement

The admission control calculations presented here depend upon the transmitting port being able to select the correct frame to transmit according to strict priority among the CQF priority levels, and initiate all transmissions in that order, without introducing extra inter-frame gap time. Since, with CQF, no buffer has frames both arriving and being transmitting at the same instant, this should pose no insurmountable problems for implementors.

4.5 Parameterization of Multi-CQF

Let us go through the exercise of initializing an input/output port pair for Multi-CQF. In the process, we will collect a set of parameters that can be used with protocols and/or network management to monitor and control the operation of Multi-CQF.

4.5.1 Cycle wander

IEEE Std 802.1Q scheduled transmission feature assumes that every bridge has a "system clock" (this term is local to this paper) that is synchronized with the other bridges' system clocks in the network, so that configuring a transmission time in one bridge has meaning in another bridge. In this paper, we do not assume synchronization of the system clocks, but we do assume frequency lock. That is, there is a maximum difference in the elapsed time between two events as measured by two bridges' system clocks, no matter the length of that elapsed time. For two bridges, this worst-case system clock difference is sysClockVar, and its units are time. We will assume that this parameter is configured by management, based on network design parameters and system data sheets. It is possible that this parameter can be adjusted during network operation. A bridge could have more than one system clock, and be connected to another system by multiple links, but there is only one value for sysClockVar for any given port, because we assume point-to-point links. We will assume that the variation can be in either direction, this-end-late or this-end-early.

IEEE Std 802.1Q scheduled transmissions and scheduled input gates are assumed to operate under control of a clock that is local to a port. In 802.1Q, the management controls that configure the schedule are defined in terms of the system clock. The bridge aligns the port clock(s) with the system clock either periodically or continuously. There is thus a worst-case excursion of the actual start of a cycle from the time configured in terms of the system clock. We parameterize this with four parameters for each port, cycleInMaxEarly, cycleInMaxLate, cycleOutMaxEarly, and cycleOutMaxLate, all measures of time. cycleIn*** is for the input gate error, cycleOut*** for the transmission error. ***Early is the worst-case for starting the cycle before the system clock time, and ***Late the worst-case for starting after the system clock time. Having both early and late parameters allows for different implementation methods for aligning the port schedule to the system clock. This parameter is computed by the bridge from knowledge of the implementation, and is constant over the lifetime of a physical connection.

4.5.2 Link delay variation

The time taken for a frame to travel from the transmitter to the receiver can vary for two reasons: the actual delay can change, due for example to temperature variations in a multi-kilometer link, and the measurement of the link delay can vary due to various clock inaccuracies. We will deal only with actual variations, not measurement variations. [1-21-0056-00-ICne-input-synchronization-for-cyclic-queueing-and-forwarding]() explains why this is possible.

4.5.3 Calculating the number of buffers required

The procedure to calculate the number of buffers needed on an output port to support one particular input port is as follows:

1) Establish a Nominal Input Cycle Start time (NICS) for the input port, and a Nominal Output Cycle Start time (NOCS) for the output port. The NICS and NOCS each repeat every T_C seconds, according to the system clock. We will assume that the offset between them is a constant (i.e., they are both driven by the same system clock).
2) Compute the earliest time, relative to the NICS, at which the first frame of a cycle can receive its IEEE Std 802.3 clause 90 timestamp. This frame is assumed to be a minimum-length frame (64 bytes plus overhead).
3) Compute the earliest time, relative to the NICS, at which a buffer on the output port must be eligible to receive the frame. This is equal to the timestamp time in bullet (2) plus the minimum time required to move the frame through the bridge to the output buffer.
4) Compute the latest time, relative to the NICS, at which the last frame of a cycle can receive its timestamp. This frame is assumed to be a minimum-length frame.
5) If the difference between the earliest timestamp and the latest timestamp is greater than or equal to the cycle time T_C, then dead time must be imposed on the transmitter, at the end of the cycle, to reduce the difference.
6) Compute the latest time, relative to the NICS, at which the last frame of a cycle can be stored into an output buffer and be ready for selection for transmission, given the worst-case forwarding delay through the bridge.
7) Convert these earliest (2) and latest (4) arrival times to times relative to the NOCS of the output port.
8) Arbitrarily label an input port NICS event $NICS_0$. Determine the latest subsequent NOCS event, which we will label $NOCS_0$, during which the earliest-arriving frame of $NICS_0$ must be stored in the output queue.
9) Determine the earliest subsequent NOCS event, which we will label $NOCS_n$, before which the latest-arriving frame from $NICS_0$ can be stored in the queue, and still be available for transmission at the start of cycle $NOCS_n$.
10) The number of cycles $NOCS_0$ through $NOCS_n$, inclusive, is the number of buffers required for the input/output port pair, B_{io}.

The number of buffers required can sometimes be reduced by:

a) Imposing a larger dead time on the transmitter feeding the input port, at the end of every cycle;
b) Altering the phase of the output port's cycle; and/or
c) Imposing implementation-specific limitations on the flows, e.g. reducing fan-in to an output port, or restricting bridging/routing features to reduce forwarding delay variation.

Finally, let us observe that large link delay variations can be accommodated by varying the above calculation. Assuming that the variations take place slowly, and that changes in relative phase between transmitter and receiver are detected using a protocol (e.g. that in new-finn-CQF-sync-method-09-21-v1), the difference between the maximum and minimum link delay can be added to the difference between the earliest- and latest- arriving frames to increase the number of buffers allocated. The phase of the input gate can be altered by small increments as the protocol detects the phase differences, without gaining or losing cycles in the transfer. Of course, the maximum adjustment made per phase adjustment event must be removed from the allocable bandwidth.

4.5.4 Initial buffer phase

The number of buffers required on an output port is the maximum required over all input ports. This may be further increased by intentional delays (2.6). When initializing an input port, a correspondence must be made between the input and output ports, so that a frame received on the input port will be stored in a particular buffer in the output port, the one that will become the transmitting buffer in the appropriate number of output cycles in the future.

The phasing between input and output ports' cycles, and thus the number of buffers in port *o* used by port *i*, is determined by the P_{io} table defined in 4.3. We compute P_{io} when initializing CQF, or when the relative phase of the input and output ports change significantly, by selecting a time *T* that coincides with the start of an input cycle on input port *i* and computing:

$$P_{io} = (X_o - S_i - B_{io} + 1) \bmod N$$

Where X_o is the identity of the transmitting buffer on output port *o* at time *T*, B_{io} is the total number of buffers required of output port *o* by input port *i* (including the transmit buffer), S_i is the value of buffer ID *S* assigned by port *i* during the input cycle starting at time *T*, and *N* is the range of S_i, the least common multiple of the number of physical buffers over all output ports.

4.6 Externally-visible Multi-CQF managed objects and protocol items

The following list includes both objects of interest to a network manager, and information elements that might be exchanged using a link-local protocol. Most items could be carried in a protocol as a check on proper configuration of adjacent ports, with varying degrees of utility for different items. Some items can only be computed by one system, and must also be known to the adjacent system. It is for further study what protocols would be used for such information transfers, or and/or whether the transfers are best accomplished using network management.

4.6.1 Cycle and priority structure managed objects

For each output port and each input port, separately, we have:

a) The cycle time of the slowest CQF priority value.
b) The priority value of the slowest CQF cycle.

For each priority level running Multi-CQF on an input port or an output port (separately), we have:

c) The layer 2 priority value
d) The number of cycles at this priority level contained within one next-lower priority value cycle.

There are other, equivalent, ways to formulate this same information. We can divorce layer 2 priority code point from importance, for example.

These parameters are not expected to change over the lifetime of a data stream. A system would not be expected to obtain this configuration information from a neighbor through a CQF-specific protocol, though exchanging this information could be done to discover of configuration errors.

4.6.2 Cycle phase managed objects

For each output port and input port, separately, we have:

a) The start time of the slowest Multi-CQF cycle, in terms of the system clock.

   This variable establishes the phase of the input or output cycle (NICS and NOCS, see 4.5.3). Typically, this variable would be managed the network administrator for output ports. For time-synchronized systems, it can be administered for input ports, as well. Alternatively, the input phase can be determined dynamically, and be read by the network administrator. Because it is in terms of the system clock, it is of no interest to neighboring systems except, perhaps, as a configuration error check for time-synchronized networks.

4.6.3 Cycle variation information

For each output port only, we have:

a) The largest offset from the nominal (system clock) NOCS event to the actual cycle start time, in the negative (actual earlier than NOCS) direction.
b) The largest offset from the nominal (system clock) NOCS event to the actual cycle start time, in the positive (actual later than NOCS) direction.

There are other ways to express the information in these two items. These values must be known to the connected input port in order for that system to compute its buffer and dead time requirements (cycleOutMaxEarly, cycleOutMaxLate in 4.5.1). This information transfer could be accomplished by means of a protocol, managed objects, or by restrictions on implementations.

4.6.4 Dead time / bandwidth balance information

There remains the balancing of conflicting goals between dead the percentage of a cycle that is available to transmit critical data streams, and the number of buffers required on the output port. Increasing the dead time can reduce the number of buffers required, and thus the end-to-end latency of a data stream, as described in 4.5.3. There are, at the very least, the following ways to make this decision:

1) Configure the output cycle phase and number of buffers to use for all bridges, in order to establish a constant per-hop delay in a network with short links. Let each system compute the dead time on each input port required to make this work, and the bandwidth available for allocation. Convey the required dead time either by protocol or by management to the transmitters, and the available bandwidth to the admission control system.
2) Configure the output cycle phase on all bridges. Configure minimum and maximum allocable bandwidth values for each CQF priority level. Let each system compute the minimum number of buffers required to meet the minimum bandwidth value, taking advantage of the maximum bandwidth value to compute a dead time value that minimizes the number of buffers required. This would be useful in a network with very long links. Convey the resultant dead time to the transmitter via protocol, and the resultant allocable bandwidth to the admission control system.
3) Using data sheet information, configure all parameters via network management. Adjust the output port cycle phasing to optimize the delay for certain specific streams.

Given that context, the following items are required by Multi-CQF:

a) Per input port, per priority level, the total dead time that must be provided by the adjacent transmitter at the end of each transmit cycle.

   There is a component of this dead time computed in this section, as well as one computed in item (5) of 4.5.3. The sum of these must be known to the adjacent transmitting port.

b) Per output port, per priority level, the total dead time that is to be provided at the end of each transmit cycle.

   This can be configured, obtained from the adjacent input node, or be a maximum of these values.

c) The allocable bandwidth for this input port and priority level.

This has three components, the minimum of the allocable bandwidth over all output ports reachable from this input port (in the input port's own system), any limitations imposed by the input port implementation, and any maximum imposed by management. Whether this is computed by, received by, or even known by the output port, or whether allocable bandwidth is the concern only of the admission control system, is an open question.

d) The allocable bandwidth for this output port and priority level.

   This can be configured, computed from the adjacent input node's requirements, or be a minimum of these values. . Whether this is computed by, received by, or even known by the output port, or whether allocable bandwidth is the concern only of the admission control system, is an open question.

5 Other issues

5.1 Frame size problem

The above discussion has largely assumed that each Stream consists of frames of a uniform size, equal to the Stream's maximum frame size. Of course, this is not always true.

The advantage of uniform frame size is that, in the ideal case, one can allocate a Stream one frame per cycle, and choose the cycle time and/or the Stream's bandwidth reservation so that there is no wasted bandwidth. Similarly, if we imagine that a Stream alternates frames of 4000 bit times and 800 bit times, we can allocate 4800 bit times per T_C and still get perfect results.

But, in a service provider situation where we are allocating a certain bandwidth per customer, but the frame sizes are essentially random, things are not so simple. Let us suppose that the maximum frame for a Stream is 13000 bit times, which is approximately equal to a maximum-length Ethernet frame, and that the cycle time T_C = 100µs. 13000/100µs = 130 Mbits/s. But, allocating a bandwidth of 13000 bits/T_C will not give the Stream 130 Mb/s. In the worst case, one 13000 bit frame followed by one minimum-length frame = 672 bits, the Stream gets (13000+672)/(200 µs) = 68.36 Mb/s.

We could overprovision the Stream by a factor of almost 2, keep the same T_C, and get minimal latency. However, we could also assign the Stream to a longer T_C. In the worst case, there are (13000–8) wasted bits in each cycle. Therefore, we can guarantee 130 Mb/s using a cycle time of 500µs by provisioning (5*13000 + 13000 – 8)/(5*100µs), or 156 Mb/s, which is a 20% overprovisioning, rather than a 90% overprovisioning, at the cost of five times the per-hop latency.

This overprovisioning/latency tradeoff is only needed for Streams that have variable frame sizes, such as service provider Streams. But, for those Streams, the lengths of the links may be a larger source of latency than the queuing delays, so the situation may not be so bad. Also, any unused bandwidth is available to non-TSN data, so overprovisioning may not be a serious concern.

5.2 Bundling

IEEE Std 802.11n combines a number of Ethernet frames into a single transmission unit, in order to minimize the number of times per second a different transmitter starts sending data. Similarly, each CQF Stream, on ingress to the TSN network, can be run through a "sausage maker". That is, frames can be encapsulated using a scheme that combines and/or splits frames into uniform-sized chunks (sausages), either small or large, that can be carried end-to-end through the TSN network, then split out into their original form. This means that overprovisioning due to the mix of frame sizes is reduced to that required by the encapsulation, itself. (In fact, that overhead can be negative, if small frames are bundled into large transmission units.)

5.3 Tailored bandwidth offerings

We can note that, in a service provider environment, overprovisioning can be almost eliminated by a combination of 1) bundling (5.2) and 2) offering the customer only a specific set of choices for a bandwidth contract, corresponding to the values of T_C implemented in the provider's network.

5.4 Overprovisioning is not always bad

Overprovisioning the bandwidth (allocating more of T_A than is necessary) is not always a bad thing:

a. Allocating a Stream to a higher priority (smaller T_C) than it needs reduces its worst-case latency. This may be necessary to meet a Stream's end-to-end latency requirement. That is, one can overprovision the frame rate in order to obtain a reduced latency. The unused bandwidth is still available for best-effort traffic. Not all TSN transmission selection schemes have this feature.
b. If the total bandwidth required by critical Streams is relatively low, using faster-than-necessary T_C values will both improve latency and reduce buffer requirements in the network. The allocated-but-unused bandwidth is still available to best-effort traffic, and thus may be of no consequence.

5.5 Relationship to current standards

Multiple instances of CQF, or for that matter, using more than a very few buffers per instance, will quickly use up the 8 buffers that IEEE Std 802.1Q-2018 provides timed output gates for. Some effort would, therefore, be required to reconcile multiple CQF with IEEE Std 802.1Q. If the idea proves useful, however, this author does not believe that they would be difficult to reconcile.

5.6 Fundamental CQF pros and cons

The obvious downside of CQF is that it requires clock synchronization and per-port time-based gating. On the other hand, CQF requires no per-Stream per-hop active state machines. A new stream can be provisioned by a network controller without any interaction between the network controller and any of the network's relay systems, except for configuring one system for ingress policing. Furthermore, calculation of the worst-case end-to-end latency is trivial, and the calculation made for one allocated stream is never affected by any other allocations or deallocations.