Cyclic Queuing and Forwarding for Latency Matching

df-finn-CQF-latency-matching-0919-v01
Norman Finn
September 21, 2019

Abstract

Cyclic Queuing and Forwarding (CQF, IEEE Std 8021Q-2018 Annex T) can be used for 1) matching the end-to-end latency of two different paths through a network, which is required when combining IEEE Std 802.1CB frame replication with IEEE Std 802.1Q resource reservation; and 2) greatly reducing the degree to which frames can be delivered out-of-order when using 802.1CB.

1 Introduction

This paper assumes the reader is reasonably familiar with the enhancements to CQF presented in df-finn-multiple-CQF-0919-v01 (http://www.ieee802.org/1/files/public/docs2019/df-finn-multiple-CQF-0919-v01.pdf). This is the second in a series of papers on useful expansions of the basic CQF idea.

As described in section 1 of df-finn-multiple-CQF-0919-v01, the present paper is dealing with the problem of providing a service characterized by bounded latency and zero congestion loss for continuous Streams in a store-and-forward environment.

2 Problem statement

The presentation cb-nfinn-seamless-issues-1015-v02 describes an issue with IEEE Std 802.1CB Frame Replication and Elimination for Reliability (FRER), when two member Streams are combined into a single Stream (frame elimination), and all three Streams have their bandwidth fixed by a Stream reservation. The reader should examine this presentation, especially slide 11, to understand the problem.

It is mentioned in cb-nfinn-seamless-issues-1015-v02 that fixing this problem requires that, when two member Streams have paths from the replication point to the elimination point, each of which has a different replication-to-elimination latency, the frames on the faster path must be delayed at the elimination point for a time sufficient to equalize the latencies of the two paths.

If one is using a queue and shaper, e.g. the Credit-based shaper (8.6.8.4 and 34 of IEEE Std 802.1Q-2018), then performing the delay to achieve latency matching is decidedly non-trivial. It is difficult because, when a frame arrives at elimination point, the receiving system does not know how long the frame has been in transit. Therefore, it does not know how long to hold it. The sequence numbers on the two paths give some clue, but there are odd cases, such as when the last frame before a transmission pause is lost. An obvious solution seems to be for each frame to carry a time stamp from replication to elimination. This, of course, would require significant standardization efforts.[1]

Section 2.6 of df-finn-multiple-CQF-0919-v01 mentions that one can use a large number of CQF buffers in order to purposely delay frames. The present paper describes specifically how CQF can solve the member Stream delay problem, and at the same time, greatly reduce the degree of out-of-order delivery produced by 802.1CB FRER.

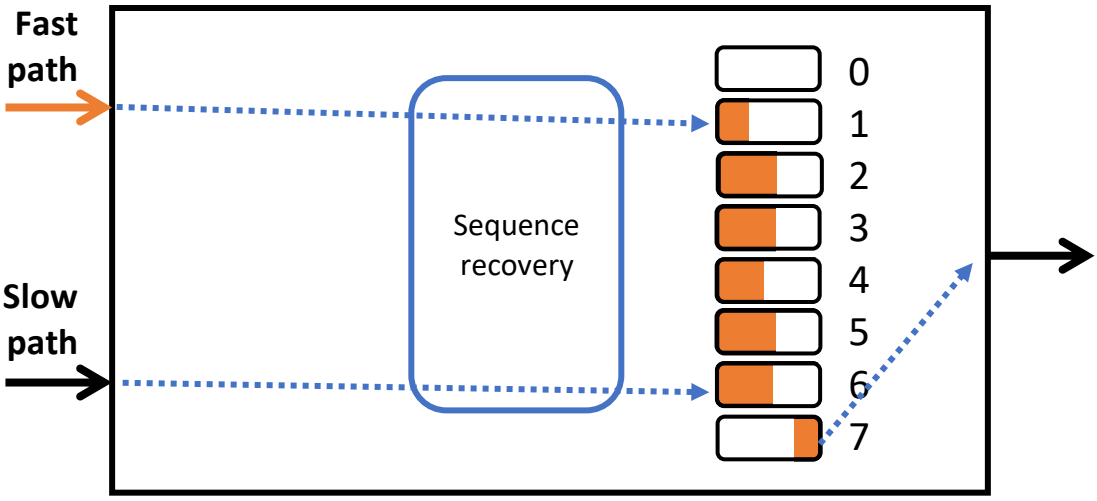
3 Delay matching

Let us assume that all of the bridges in the network use CQF. Then, the difference in latency along any two different paths in the network is in integer multiple of the CQF cycle time. Therefore, the problem of adding latency to the fast path to make its latency match the slow path reduces to buffering the fast-path frames for an additional number of CQF cycles, where the number of cycles can be obtained from the Stream reservation protocol (IEEE Std 802.1Q MSRP or IEEE P802.1Qdd RAP) or from the network controller.

In Figure 1 we see an example of a bridge that is performing the 802.1CB stream merge and sequence recovery functions (frame elimination), combining two member streams into a single stream. The two member streams are entering from the fast path (shorter, less delay) on one port, and from the slow path (longer, more delayed) on another port. The frames remaining, after the sequence recovery function eliminates the duplicates, are transmitted on the output port, on the right.

[1] This is not to say that there are not better solutions, just that the author is not aware of them.

Figure 1 CQF latency matching



At the moment in time captured in Figure 1, buffer 0 is empty, its contents having been transmitted on the previous cycle. Buffer 1 is filling from the fast path. The sequence recovery function drops no frames, because the fast-path frames have not been seen, before. In buffers 2-5, the fast frames are aging for the required number of cycle times (five, in this example). Note that buffer 4 is less full than the others; perhaps a frame was lost on the fast path. The slow path frames are being deposited into buffer 5. Very few are stored; only frames that were missing from the fast path pass through the sequence recovery function. Buffer 7 is being transmitted.

CQF latency matching requires that the frame replication node transfers the frames, from its one input port to its two output ports, using CQF. This ensures that, in the replication node, the same set of frames is in each of the CQF buffers being transmitted simultaneously on the two output ports. This is exactly what one would expect from CQF.

Let us assume that the frames on both paths were, then, regulated at every hop by CQF, running at the same cycle time since frame replication. In that case, at the elimination node shown in Figure 1, *the replacement frames deposited from the slow path go into the same CQF buffer into which those corresponding, lost, frames, would have been deposited from the fast path*.

Thus, on the output port, there is never an excess of frames in the CQF buffer. The output bandwidth reservation is never exceeded. The latency matching problem is solved.

4 Ordered delivery

If each CQF buffer is implemented as a FIFO, then the replacement frames are placed into the CQF buffer after all of the fast-path frames, and so can be delivered out-of-order. However,

they are, at most, only one cycle time out of order, not five, which eases the resequencing burden for the receiver. Of course, the CQF buffer does not have to be a FIFO. In that case, we see that the range over which the re-ordering sort function must operate is limited to a single CQF buffer, instead of the whole of the buffered fast-path stream.