9. Transmit state machines (proposal 1)

NOTE—Multiple bunch-avoiding pacing protocols are presented for consideration:
a) Clause 9 (this clause) presents a pseudo-synchronous transmission model.
b) Clause 10 presents a cross-flow shaper transmission model.

9.1 Pacing overview

9.1.1 Delays

The preferred topologies consists entirely of paced bridges, as illustrated in Figure 9.1a. Within such topologies, a frame transmitted by station a0 in cycle[*n*] incurs fixed nominal delays while passing through bridges. Thus, this frame nominally departs bridgeB in cycle[*n*+2], bridge C in cycle[*n*+4], and bridgeE in cycle[*n*+6].

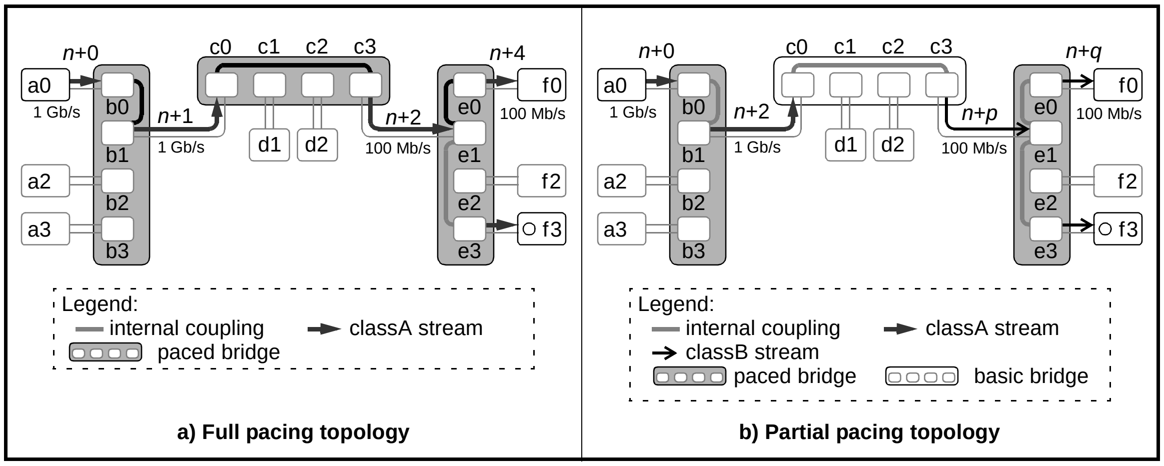

Figure 9.1—Topology-dependent pacing delays

Within Figure 9.1a, the actual transmission times can vary from their nominal targets, due to contention with other traffic. Each bridge compensates for early and late arrivals, so that the extent of deviations from nominal on link b1-to-c0 are the same as those on link e0-to-f0.

Within Figure 9.1b, an intermediate basic bridge is assumed. Output from bridgeC is therefore downgraded from classA to classB, to avoid degradation of well-paced traffic. Thus, the fully-paced properties of bridgeE still apply to possible f3-to-f0 traffic (not illustrated).

The uncertainty of cycle *p* and *q* cycle delays in Figure 9.1b are due to passing through the non-paced bridgeC. Although much of this traffic would arrive earlier, some of the traffic could be delayed up to the nominal delays of Figure 9.1a. In more complex topologies, such delays could exceed the nominal delays through paced bridges, due to bunching effects (see Annex F).

To support such topology, this working paper mandates that compliant end stations provide larger elasticity buffers (see TBD) than required within fully paced topologies. However, defining topology restrictions to ensure elasticity-buffer sufficiency is beyond the scope of this working paper.

9.1.2 Paced 1 Gb/s classA flows

Pacing involves sending accumulated classA traffic once every isochronous cycle, rather than allowing larger (typically an MTU) frames to be accumulated. After each cycle's classA traffic has been sent, the remaining time is available for sending classB/classC traffic. This provides low-jitter bandwidth guarantees, as does time division multiplexing (TDM), while allowing unused classA bandwidths to be utilized by classB/classC traffic.

A pacing bridge maintains this pacing behavior, thus avoiding problems normally associated with bunching (see Annex F). For a bridge between 1 Gb/s link1 and 1 Gb/s link2 (see Figure 9.2a), paced frames can be forwarded with a nominal 1-cycle delay (see Figure 9.2b). The 1-cycle delay is necessary to account for offset migration and store-and-forward processing delays.

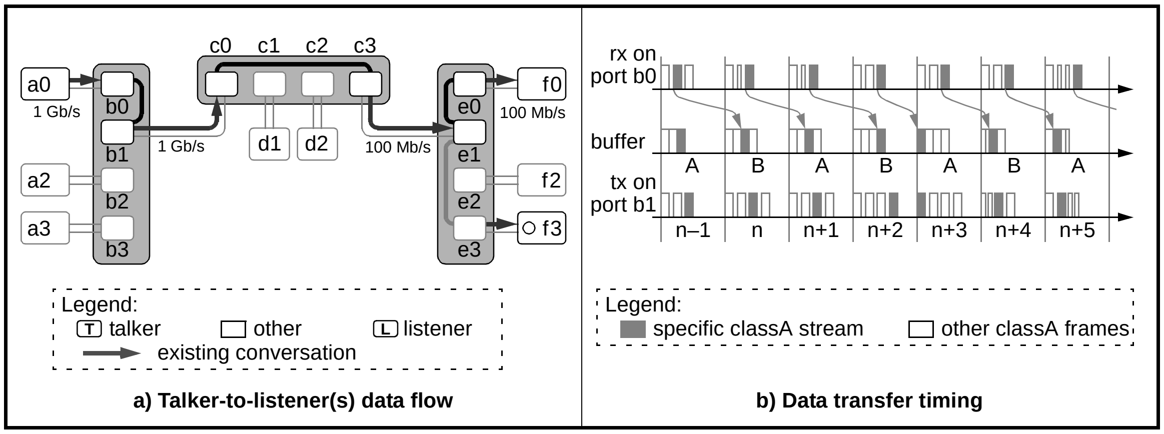

Figure 9.2—Paced 1 Gb/s classA flows

Offset migration refers to changes in a classA frame's within-cycle placement on (for example) link1 and link2. Depending on the timing of unrelated events, the offset of the classA-data frame within the cycle can migrate over time, as other conversations are started, ended, advanced, delayed, joined, or routed elsewhere.

A possible implementation could utilize double output buffers, processed as follows:

cycle[n+2×k+0]: classA traffic is saved in buffer[A] and transmitted from buffer[B].
cycle[n+2×k+1]: classA traffic is saved in buffer[B] and transmitted from buffer[A].

The boundaries between cycles are marked by a distinct set of cycleSync markers (not illustrated), rather than relying on precise time-synchronization and deadbands to imply their temporal placement.

The classA transmissions within each cycle are shaped, to allow for unrelated asynchronous frame transmissions. The shaper allows a higher-than 75% transmission rate, to ensure transmission completion well before before the next cycle begins, even in the presence of conflicting non-classA transmissions.

To better understand the minimal buffer requirements, consider frame transfers that are momentarily disrupted by an MTU-sized classC transmission, started near the end of link1's classA transmissions. For the receive-side slippage scenario of 9.3a, data[*n*] arrives in cycle[*n*] and fills buffer[A]. Since buffer A is not destined for transmission until cycle[*n*+1], conflicts are avoided.

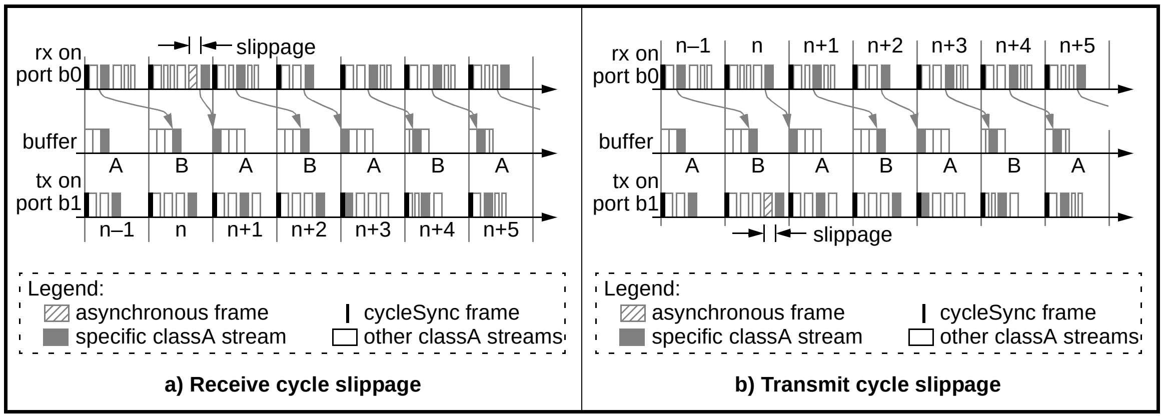

Figure 9.3—Cycle slippage

For the transmit-side slippage scenario of Figure 9.3b, buffer[B] is fully emptied in cycle[*n*]. Since buffer[B] is not destined for filling until cycle *n*+1, conflicts are avoided.

9.1.3 Paced 100 Mb/s flows

Editors' Notes: To be removed prior to final publication.
A two-cycle delay is illustrated, although the protocols can be simplified by assuming a three cycle delay. The tradeoff between protocol simplicity and a passthrough latency has not been carefully reviewed.

A 100 Mb/s pacing bridge also maintains this pacing behavior, thus avoiding problems normally associated with bunching (see Annex F). For a bridge between 100 Mb/s link3 and 100 Mb/s link4 (see Figure 9.4a), paced frames can be forwarded with a nominal 2-cycle delay (see Figure 9.4b).

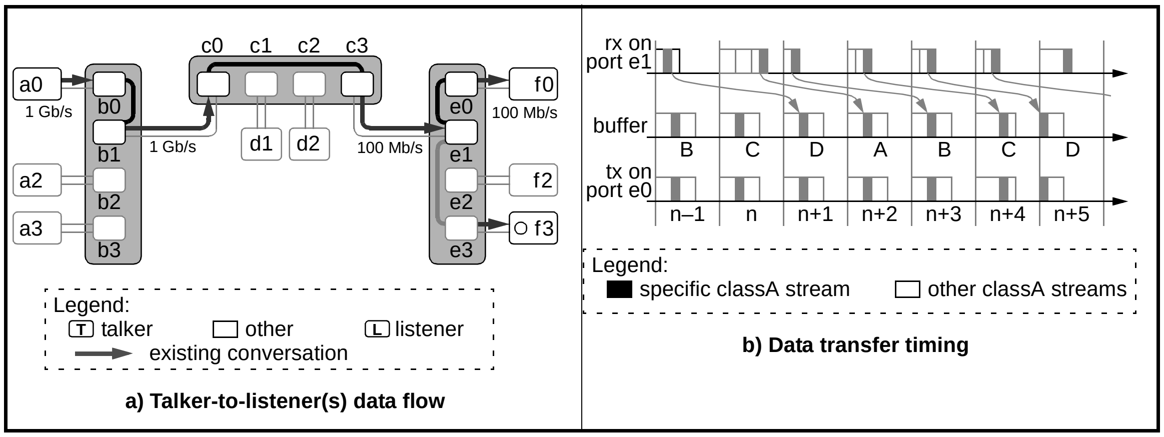

Figure 9.4—Paced 100 Mb/s classA flows

A possible implementation would involved six output buffers, processed as follows:

cycle[n+4×k+0]: classA traffic is saved in buffer[A] and transmitted from buffer[C].
cycle[n+4×k+1]: classA traffic is saved in buffer[B] and transmitted from buffer[D].
cycle[n+4×k+2]: classA traffic is saved in buffer[C] and transmitted from buffer[A].
cycle[n+4×k+3]: classA traffic is saved in buffer[D] and transmitted from buffer[B].

To better understand the minimal buffer requirements, consider frame transfers that are momentarily disrupted by an MTU-sized classC transmission, started near the end of link3 classA transmissions. For the receive-side slippage scenario of Figure 9.5a, data[n] arrives in cycle[n+1] and fills buffer[A]. Since buffer A is not destined for transmission until cycle[n+2], conflicts are avoided.

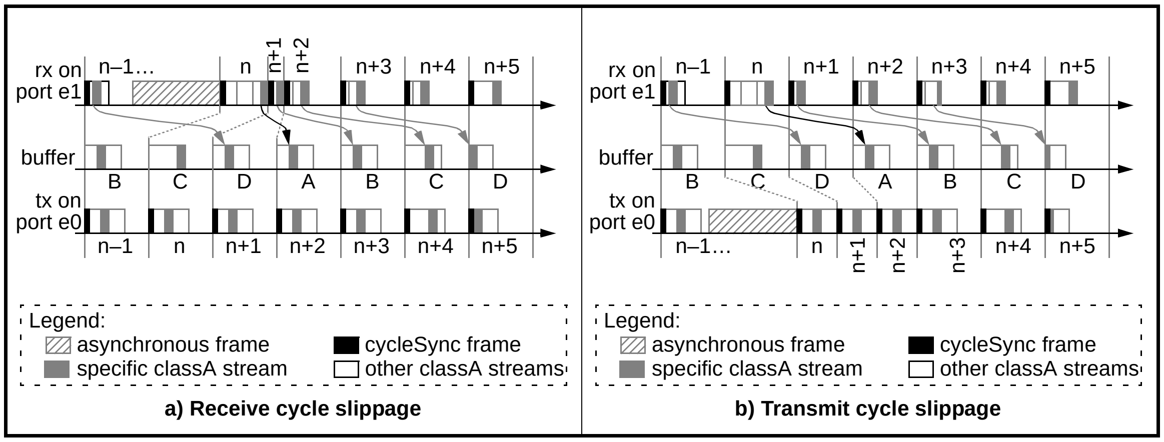

Figure 9.5—Cycle slippage

For the transmit-side slippage scenario of Figure 9.5b, buffer[D] is fully emptied in cycle[n+1] and in cycle[n+2]. Since buffer[D] is not destined for filling until cycle n+3, conflicts are avoided.

To achieve a robust 2-cycle latency objective, restrictions are placed on non-classA transmissions. These restrictions are as follows:

a) An MTU (or sequence of frames not exceeding an MTU) may be appended to the last classA frame within any cycle whose cycleSync frame transmission was not delayed.

b) Within any cycle, any non-classA frame may be transmitted after the last classA frame, but only if this frame transmission would not delay the transmission of the next cycleSync frame.

Condition (a) is sufficient to ensure that all transmissions occur within the intended or following cycle, assuming a 100 Mb/s span, 2000 byte MTU, 125 µs cycle, and 75% classA loading. With these assumptions, the worst-case delay from the start of the intended cycle, as specified by Equation 9.1, is well within the 2-cycle 250 µs constraint.

$$delay \ge (\text{MTU} - 0.25 \times \text{cycle}) + 0.75 \times \text{cycle} \quad (9.1)$$
$$delay \ge 2\,000 \times ((8\text{ bits/byte}) \times (1\text{ second})/(100\text{ Mb/s})) + 0.50 \times (125\ \mu\text{s})$$
$$delay \ge (160\ \mu\text{s}) + (62.5\ \mu\text{s})$$
$$delay \ge 222.5\ \mu\text{s}$$

9.1.4 Transmit port structure

An end station and bridge have functionally distinct transmit queues for classA, classB, and classC traffic, allowing each to be managed separately, as illustrated in Figure 9.6. The transmit port is responsible for pacing classA/classB traffic and shaping classB/classC traffic, so as to limit the high-class traffic to 75% of the link bandwidth. The transmit-port structure is slightly different for 100 Mb/s and 1 Gb/s transmit ports, due to the distinct times associated with an MTU transmission.

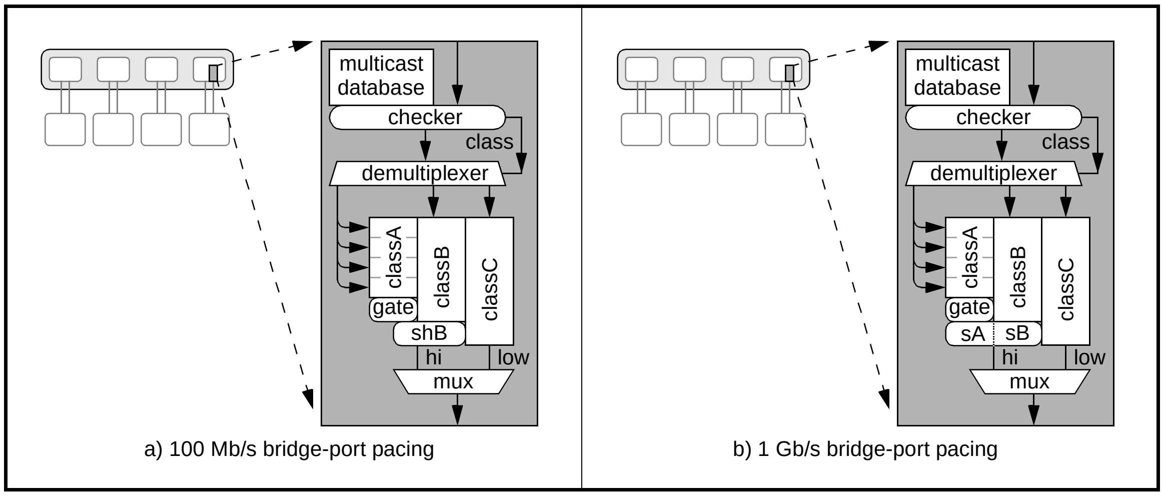

Figure 9.6—Transmit-port structure

Although classA frames have the highest priority, the classA frames are gated to prevent their early departure. Gating involves blocking classA frames that arrived with *sourceCycle*= *n*, until the start of cycle *n*+*p*. After the start of cycle *n* +*p*, the transmitter waits for the completion of preceding non-classA frames (or residual cycle *n*+*p*–1 classA frames), then transmits these arrived-in-cycle-*n* frames with *sourceCycle*= *n*+*p*. As noted previously, *p* is a design-dependent integer constant, preferably no more than 4 cycles (see 5.1.2 and 5.1.3).

A bridge has to cope with frame-reception uncertainties (due to preceding frame-transmission uncertainties), in addition to its own frame-transmission uncertainties. As such, the values of *p* are expected to be slightly larger in bridges than in talker-station or listener-station designs.

Within bridges, the distinction between service classes is based on the multicast addresses within frames. These multicast addresses are checked against the multicast database, which supplies *class* information in addition to the normal multicast routing (forward or not-forward) information. This *class* information controls the demultiplexer, which routes to the appropriate classA, classB, or classC output queues.

The cycle slippage on a 100 Mb/s link mandates the use of four 3/4-cycle output buffers, which incur a 2-cycle pass-through delay. The classA traffic is gated to avoid wrong-cycle transmissions and excessive consumption, but is not otherwise not shaped. The overlapping shB shaper of Figure 9.6a is intended to illustrate the use of classA transmission counts and the classB shaper, not the shaping of classA traffic.

On such 1 Gb/s transmitter ports, the classA traffic is shaped to reduce lower-class blockage, as well as gated to avoid wrong-cycle transmissions and excessive consumption. The adjacency of shA/shB shapers in Figure 9.6b is intended to illustrate distinct classA/classB shaping functions, but sharing of classA transmission counts between shapers.

Achievable delays through a bridge depend only on the speed of the input-link speed, as summarized in Table 9.1. These numbers are slightly misleading, since transmissions on a 100 Mb/s link have implied additional delays incurred when passing through its adjacent 100 Mb/s receiver.

Table 9.1—ClockPort state table

Link type		Delay	
Input	**Output**	**Cycles**	**Time**
100 Mb/s	—	2	250 µs
1 Gb/s	—	1	125 µs

9.1.5 Pacing at 1 Gb/s

Pacing at 1 Gb/s, as illustrated in Figure 9.7. For ontime cycles, a residual amount of classB/classC traffic is allowed throughout the cycle, as illustrated in Figure 9.7a. For slipped cycles, a residual amount of classB/classC traffic becomes available after the delay effects have been overcome, as illustrated in Figure 9.7b.

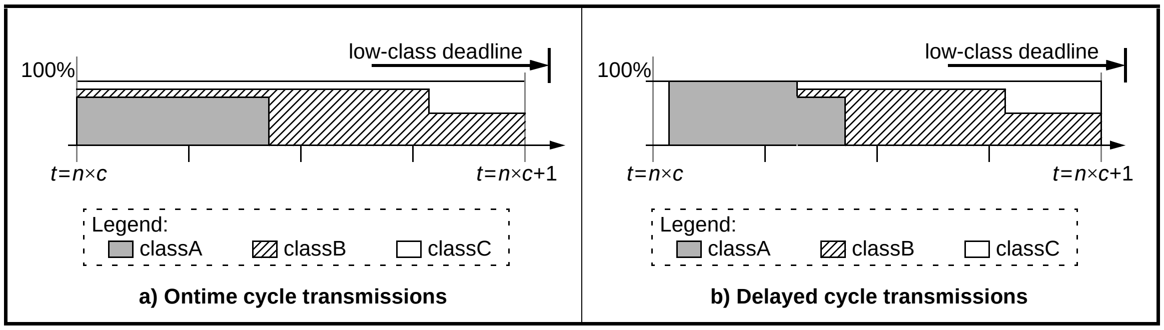

Figure 9.7—Pacing at 1 Gb/s

9.1.6 Pacing at 100 Mb/s

Pacing at 100 Mb/s, as illustrated in Figure 9.8. For ontime cycles, a residual amount of classB/classC traffic is allowed throughout the cycle, as illustrated in Figure 9.8a. For delayed cycles, a residual amount of classB/classC traffic becomes available after the delay effects have been overcome, as illustrated in Figure 9.8b.

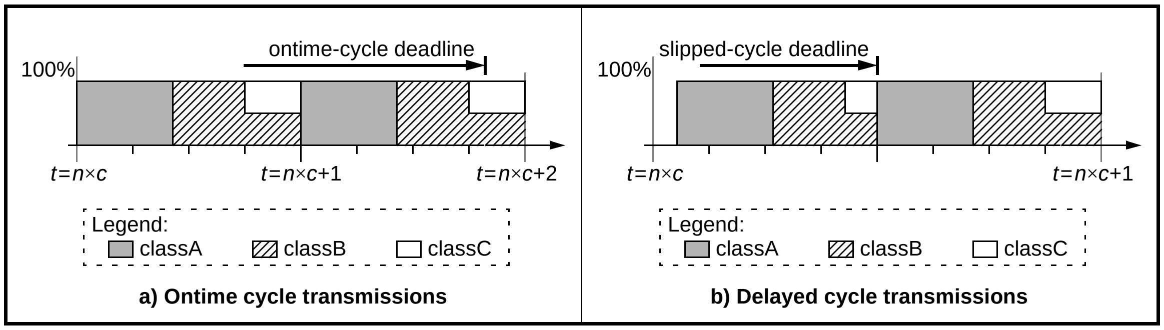

Figure 9.8—Pacing at 100 Mb/s

9.1.7 Shaper behavior

Although multiple shaper are specified within this working paper, the behavior of most shapers can be characterized by a common algorithm and instance-specific parameters (as done within RPR[B5]). The shapers' credits are adjusted down or up, as illustrated in Figure 9.9. The decrement and increment values typically represent sizes of a transmitted frame and of credit increments in each update interval, respectively.

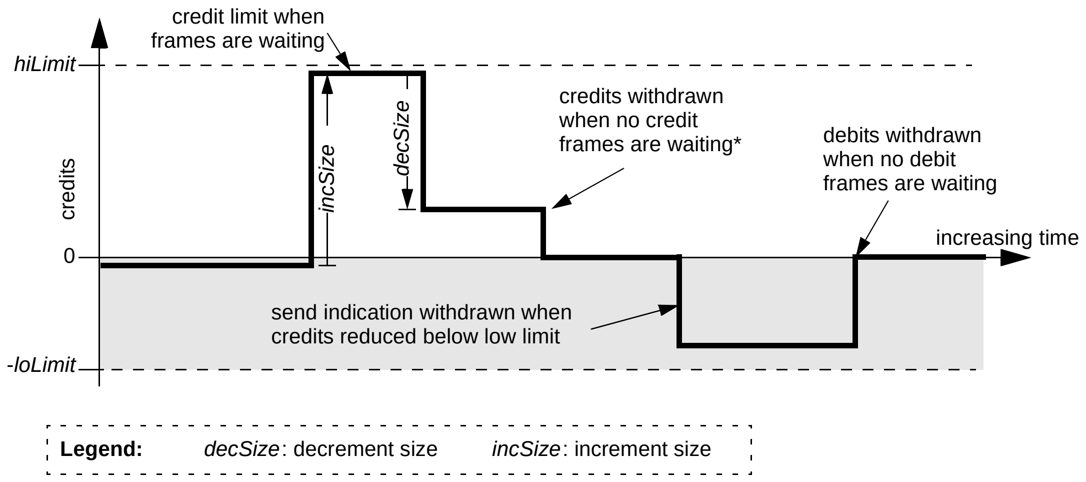
credit limit when frames are waiting
hiLimit
credits withdrawn when no credit frames are waiting*
debits withdrawn when no debit frames are waiting
incSize
decSize
credits
increasing time
0
send indication withdrawn when credits reduced below low limit
-loLimit
Legend: decSize: decrement size incSize: increment size

Figure 9.9—Credit adjustments over time

Crossing below the zero threshold generates a rate-limiting indication (the removal of a send indication), so that offered traffic can stop. By design, the credit value never goes below the –*loLimit* extreme. To bound the burst traffic after inactivity intervals, when no frames are ready for transmission, credits are reduced to zero (if currently higher than zero) and can accumulate to no more than the zero-value limit.

The *hiLimit* threshold limits the positive credits, to avoid overflow. When frames are ready for transmission (and are being blocked by transit traffic), credits can accumulate to no more than this *hiLimit* value.

In concept, the shapers consist of a token bucket. The credits in the token bucket are incremented by the size of each debit-frame when it is being transmitted. The number of credits in a token bucket is decremented by the size of each credit-frame when it is being transmitted. When a credit-frame is waiting, it is transmitted only if the number of credits in the token bucket is positive; When a debit-frame is waiting, it is transmitted only if the number of credits in the token bucket is negative.

9.2 Terminology and variables

9.2.1 Common state machine definitions

The following state machine inputs are used multiple times within this clause.

queue values
  Enumerated values used to specify shared queue structures.
    QP_TX_PUSH—The input port's receive-from-ports queue.
    QP_TX_CA—The first of the output port's classA buffers.
    QP_TX_CB—The output port's classB queue.
    QP_TX_CC—The output port's classC queue.
    QP_TX_LINK—The output port's transmit-PHY queue.
    QP_TX_SYNC—The port's queue that provides clockSync frames.

9.2.2 Common state machine variables

One instance of each variable specified in this clause exists in each port, unless otherwise noted.

currentTime
  A value representing the current time.
mtuSize
  The size of the maximum transfer unit (MTU).
    Value: 2000 bytes

NOTE—The specified *mtuSize* is larger than currently supported by IEEE Std 802.3, but consistent with expected near-term frame-extension revisions of this standard.

speedIs100Mbs
  A value that communicates the operating speed of the link.
    TRUE—The port is operating at a speed of 100 Mb/s.
    FALSE—The port is operating at speeds of 1 Gb/s or above.
thisCycle
  A cycle counter derived from *thisTime*, as defined by Equation 9.2.
```
    Floor(thisTime * 8000);                                    (9.2)
```
thisTime
  A normalized time-of-day counter derived from *timeOfDay*, as defined by Equation 9.3.
```
    (timeOfDay / (4.0 * (1<<30)))                              (9.3)
```

9.2.3 Common state machine routines

–none–

9.2.4 Routines defined in other clauses

This clause references the following routines defined in Clause 7:

Dequeue(*queue*)
Enqueue(*queue*, *frame*)
Min(*value1*, *value2*)
    See 7.2.3.

9.3 Pacing state machines

9.3.1 ReceiveRx state machine

The ReceiveRx state machine is responsible for receiving pacing classA traffic, shaped classB traffic, and best-effort classC traffic. An intent is to transfer each to the appropriate output queue.

The following subclauses describe parameters used within the context of this state machine.

9.3.1.1 ReceiveRx state machine definitions

CYCLE_SYNC
    An assigned *subType* value that distinguishes a clockSync from other Residential Ethernet frames.
GROUP_BIT
    A constant value derived from IEEE Std 802-2001 and specified by Equation 9.4.
    `((macAddress & GROUP_BIT) != 0)` (9.4)

queue values
    Enumerated values used to specify shared queue structures.
        QP_TX_CA, QP_TX_CB, QP_TX_CC
        QP_TX_PUSH
            See 9.2.2.
RES_ETHER
    The *protocolType* code value assigned to Residential Ethernet.

9.3.1.2 ReceiveRx state machine variables

class
    A value that represents the results of a forwarding database search.
delta
    A value that represents the difference between frame-signaled and computed cycle values.
frame
    The contents of a received frame.
myCycle
    The two least-significant bits of the *thisCycle* value.
queueA
    The selected classA queue identifier, based on *delta*-selected locations.
speedIs100Mbs
thisCycle

thisTime
See 9.2.2.

9.3.1.3 ReceiveRx state machine routines

DataBaseClass(*macAddress*, *port*)
Provides a forwarding database indication of how the *macAddress* is routed to the specified *port*.
CLASS_A—The associated multicast frame is forwarded as classA traffic.
CLASS_B—The associated multicast frame is forwarded as classB traffic.
CLASS_C—The associated multicast frame is forwarded as classC traffic.
BLOCKED—The associated multicast frame is not forwarded.
Dequeue(*queue*)
See 9.2.4.
EnqueuePort(*port, queue, frame*)
Places the *frame* at the tail of the specified *queue* within the specified *port*.
ForwardUnicast(*frame*)
Forwards a unicast frame to the selected output port, if any.
This routine mimics existing standards, which remain unaffected by this working paper.
Multicast(*macAddress*)
Indicates whether the supplied address is a multicast (or broadcast) address, as specified by Equation 9.5.
TRUE—The address is a multicast (or broadcast) address.
FALSE—(Otherwise.)

```
((macAddress & GROUP_BIT) != 0)
```
(9.5)

9.3.1.4 ReceiveRx state table

The ReceiveRx state machine is specified in Table 9.2. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 9.2—ReceiveRx state table

Current		Row	Next	
state	**condition**		**action**	**state**
START	(frame = Dequeue(QP_TX_PUSH)) != NULL	1	—	FIRST

	—	2	myCycle = (thisCycle % 4);
			delta = (4 + myCycle – rxCycle) % 4;
				PLACE

PLACE	delta == 3	1	delta = 0;	PLUS
	—	2	—	
PLUS	speedIs100Mbs	1	queueA = QP_TX_CA + (4 + 2 – delta) % 4;	START
	—	2	queueA = QP_TX_CA + (4 + 1 – delta) % 4;	
FIRST	frame.protocolType==RES_ETHER && frame.subType == CYCLE_SYNC	1	rxCycle = (frame.cycleCount % 4);	START
	Multicast(frame.da)	2	class = DataBaseClass(frame.da, port);	CAST
	—	3	ForwardUnicast(frame)	START
PUSH	class == CLASS_A	1	EnqueuePort(port, queueA, frame);	START
	class == CLASS_B	2	EnqueuePort(port, QP_TX_CB, frame);	
	class == CLASS_C	3	EnqueuePort(port, QP_TX_CC, frame);	
	—	4	—	

Row START-1: If a frame has arrived, process that frame.
Row START-2: Otherwise, compute the cycle offset for later classA queue placement.

Row PLACE-1: Frames that arrive early are processed as though they arrived within this cycle.
Row PLACE-2: Otherwise, the difference between labeled and actual cycles determines frame placement.

Row PLUS-1: Frames arriving from a 100 Mb/s link are placed 2-cycles ahead, to allow for cycle slips.
Row PLUS-2: Frames arriving from a 1 Gb/s link are placed 1-cycle ahead, since cycle slips are avoided.

Row FIRST-1: The cycleSync frames identify the cycle number, despite cycle-slip possibilities.
Row FIRST-2: Multicast frames are sent to all enabled ports.
Row FIRST-3: Unicast frames are processed normally.

Row PUSH-1: Multicast classA frames are forwarded to the appropriate cycle-sensitive classA queue.
Row PUSH-2: Multicast classB frames are forwarded to the classB queue.
Row PUSH-3: Multicast classC frames are forwarded to the classC queue.
Row PUSH-4: If no class is specified, multicast frames are not routed through this port.

9.3.2 TransmitTx state machine

The TransmitTx state machine is responsible for pacing/shaping classA traffic and shaping classB traffic destined for 1 Gb/s links. An intent is to support projected MTU-sized transfers and interleaved lower-class traffic, without exceeding the 1-cycle delay inherent with cycle-synchronous bridge-forwarding protocols.

The following subclauses describe parameters used within the context of this state machine.

9.3.2.1 TransmitTx state machine definitions

BPS

Represents a bound on the number of transmitted bytes per second, as defined by Equation 9.6.

```
    (speedIs100Mbs ? 12500000 : 125000000)                          (9.6)
```

CAP

Represents a bound on the number of transmitted bytes, as defined by Equation 9.7.

```
    ((speedIs100Mbs && phase != MORE) ?                             (9.7)
      ((cycle + 1) * 8000. - thisTime) * BPS : MTU)
```

queue values

Enumerated values used to specify shared queue structures.

QP_TX_CA
QP_TX_CB
QP_TX_CC
QP_TX_LINK
QP_TX_SYNC

See 9.2.2.

9.3.2.2 TransmitTx state machine variables

creditA

A shaper credit whose positive value enables classA/classB primary transmissions.

creditB

A shaper credit whose positive/negative values enable secondary classB/classC transmissions.

cycle

The cycle whose classA data is being transmitted.

cycleSize

The number of bytes included within a 125 µs cycle.

Value:

1562.5—for 100 Mb/s links
15625—for 1 Gb/s links

frame

The contents of a to-be-transmitted frame.

hiLimitB

A value that limits the cumulative *creditB* credits.

Value: MTU.

limit

A value that limits the amount of transmitted primary classA/classB bandwidth.

loLimitB

A value that limits the cumulative *creditB* debits.

Value: MTU.

phase

An indication of what remains to be transferred within the cycle.

HEAD—The cycleSync frame are to be sent.
MORE—Other classA/classB frames are to be sent.
DONE—All classA frames have been sent.

queue
A variable that identifies the appropriate classA queue for this cycle's transmissions.
speedIs100Mbs
See 9.2.2.
thisCycle
thisTime
See 9.2.2.

9.3.2.3 TransmitTx state machine routines

Cap(*speedIs100Mbs*, *phase*, *creditA*, *cycle*, *thisTime*)
Provides a cap on the lengths of classB and classC transmissions.

```
if (speedIs100Mbs) {                                    (9.8)
  if (phase == MORE)
    return(0);
  near = (cycle + 1.0) * 8000;
  safe = (cycle + 0.8) * 8000;
  return(thisTime <= safe ? MTU : near * BPS);
} else {
  if (phase == MORE)
    return(-creditsA/16);
  near = (cycle + 1.05) * 8000;
  return((near - thisTime) * BPS);
}
```

Dequeue(*queue*)
See 9.2.4.
DequeueSize(*queue*, *size*)
Returns the next available frame from the specified queue, from frames no larger than *size*.
Enqueue(*queue*, *frame*)
See 9.2.4.
QueueEmpty(*queue*)
Returns the an indication of whether the queue is empty.
0—The specified queue is not empty.
1—The specified queue is empty.
Size(*frame*)
Returns the size of the specified frame.

9.3.2.4 TransmitTx state table

The TransmitTx state machine is specified in Table 9.3. The link-speed independent rows are white; the link-speed dependent rows are shaded. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 9.3—TransmitTx state table

Current		Row	Next	
state	**condition**		**action**	**state**

START	cycle > (thisCycle + 1)	1	cycle = thisCycle;
			phase = HEAD;
				PREP

	cycle < (thisCycle – 1)	2		

	cycle < thisCycle && phase == DONE	3	cycle += 1;
			phase = HEAD;	

	!QueueEmpty(QP_TX_LINK)	4	—	START

	phase == HEAD	5	queue = QP_TX_CA + (cycle % 4);
			frame = Dequeue(QP_TX_SYNC);
			limit = 0.75 * cycleSize;
			creditA=
			16 * BPS * (thisTime – cycle*8000.);
			phase = MORE;
				POST

	—	6	cap = Cap(speedIs100Mbs,
			phase, creditA, cycle, thisTime);
				PLUS

PLUS	creditB >= 0 && (frame =
	DequeueSize(QP_TX_CB), cap) != NULL
		1	creditB =
			Max(loLimitB, creditB – Size(frame));
			creditA += 16 * Size(frame);
				FINAL

	creditB <= 0 && (frame =
	DequeueSize(QP_TX_CC, cap)) != NULL
		2	creditB =
			Min(hiLimitB, creditB +Size(frame));
			creditA += 16 * Size(frame);	

	(frame = DequeueSize(QP_TX_CB, cap)) != NULL	3	creditA += 16 * Size(frame);	
	(frame = DequeueSize(QP_TX_CC, cap)) != NULL	4		
	phase != MORE	5	—	START
	(frame = DequeueSize(queue, limit)) != NULL	6	—	POST
	(frame = Dequeue(queue)) != NULL	7	—	START
	(frame = DequeueSize(QP_TX_CB, limit)) != NULL	8	limit –= Size(frame);	FINAL

	—	9	phase = DONE;
			creditB= Min(hiLimitB, limit+creditB);
				START

POST	—	1	limit –= Size(frame);
			creditA –= Size(frame);
				FINAL

FINAL	—	1	Enqueue(QP_TX_LINK, frame);	START

Row START-1: If *cycle* has advanced two-beyond *thisCycle*, something is in error.
(The *cycle* value can advance one-beyond *thisCycle*, due to small *timeOfDay* update discontinuities.)
Row START-2: If *cycle* has dropped two-behind *thisCycle*, something is in error.
(Large *timeOfDay* update discontinuities can cause cycle to advance or retreat beyond normal bounds.)
|**Row START-3:** The phase is initialized to HEAD at the start of each cycle.
Row START-4: Wait for the queue to be emptied, so something can be transmitted.
Row START-5: When the next cycle starts, a clockSync frame is transmitted.
The *limit* value is set to limit classA transmissions to no more than 75% of the link bandwidth.
The *creditA* value initialized to account for cycleSync frame slippage (for 1 Gb/s ports only).
Row START-6: Set caps on the maximum transmission size of classB/classC transmissions.

Row PLUS-1: If enabled and available, a classB frame is transmitted.
The *creditB* values is decremented by the transmitted frame size, to effect a classB shaper.
Row PLUS-2: If enabled and available, a classC frame is transmitted.
The *creditB* values is incremented by the transmitted frame size, to effect a classB shaper.
Row PLUS-3: If available, a classB frame is transmitted.
Row PLUS-4: If available, a classC frame is transmitted.
Row PLUS-5: Otherwise, no frame is transmitted.
Row PLUS-6: An enabled, available, and properly sized classA frame is readied for transmission.
Row PLUS-7: An enabled, available, and improperly sized classA frame is discarded.
Row PLUS-8: An enabled, available, and properly sized classB frame is readied for transmission.
Row PLUS-9: If enabled but unavailable, this cycle’s primary frame transmissions have completed.

Row POST-1: The shaper’s creditA value is decremented to lightly throttle primary transmissions.
The limit value is also decremented, to enforce the 75% cycle classA/classB transmission limitation.

Row FINAL-1: Transmission is affected by placing the frame in the port’s transmit queue.