10. Transmit state machines (proposal 2)

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

10.1 Rate-based scheduling overview

The clause describes a rate-based scheduling technique. The rate-based scheduling concepts are similar to those within rate monotonic scheduling protocols, commonly used within real-time systems. Objectives associated with time-sensitive forwarding alternatives include the following:

a) Multiple time-sensitive transmission rates are supported, including:
   1) Highest rate 8 kHz traffic, such as the traffic generated by simple bridges between RE and existing IEEE 1394[B6] A/V devices.
   2) Lower rate traffic, such as voice over internet protocol (VOIP) traffic, without forcing this traffic to be reblocked into smaller (and therefore less efficient) frame sizes.

b) Frame forwarding should not be dependent on successful time-of-day synchronization between the bridge and adjacent stations. Frame forwarding should succeed before the grand clock-master station has been selected, or when the selected grand-master clock station changes.

c) Frame-forwarding protocols should leverage existing bridge queue and service models, although specification of abstract rate shaper details is expected.

d) Each traffic class has guaranteed latency, when the cumulative traffic is constrained to less than the link capacity. The latency guarantee is approximately an MTU more than an inter-arrival period.

Rate-based scheduling involves associating a priority with frame transmissions, where the priority is a monotonic function of the frame transmission frequency, as illustrated in Figure 10.1.

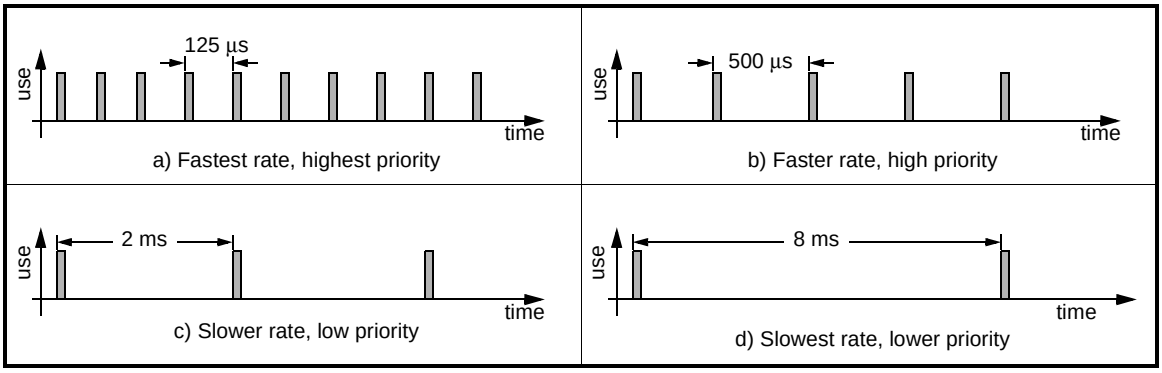

Figure 10.1—Rate-based priorities

An application could desire to send traffic once in every 8 ms interval, while attempting to constrain the worst-case latency to a smaller 2 ms value, as illustrated in Figure 10.2-a. This would be done by subscribing for a 2 ms interval, while only sending the traffic in larger 8 ms intervals, as illustrated in Figure 10.2-b. The lower latency is not achieved without costs: the effective subscription bandwidth allocated to this application is 4 times larger than necessary.

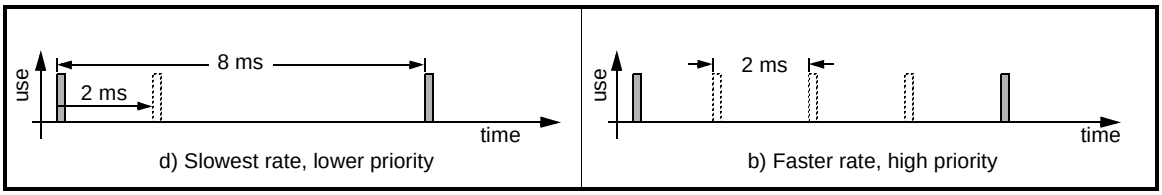

Figure 10.2—Rate-based priorities

10.1.1 Rate-based priorities

Quality of service is based on the availability of user_priority field parameter associated within transmitted time-sensitive frames, as listed in Table 10.1.

Table 10.1—Tagged priority values

Code	Interval (ms)	Name	Description
0	n/a	CLASS_C	Best effort, with minimal guaranteed BW
1	n/a	CLASS_B	Preferred, with minimal guaranteed BW
2-3	—	—	Used for other purposes
4	8	CLASS_A3	Guaranteed BW over longer interval
5	2	CLASS_A2	Guaranteed BW over long interval
6	0.5	CLASS_A1	Guaranteed BW over short interval
7	0.125	CLASS_A0	Guaranteed BW over shorter interval

10.1.2 Port-to-port reshaping, without bunching accumulation

The concept of rate-based scheduling assumes shaped talkers and reshaped talker agents within bridges, as illustrated in Figure 10.3 (only the components associated with specific flows are illustrated). From a functional perspective, multiple shapers are required per class, where each class is illustrated as a distinct layer. From a practical perspective, one shaper instance with multiple contexts is sufficient.

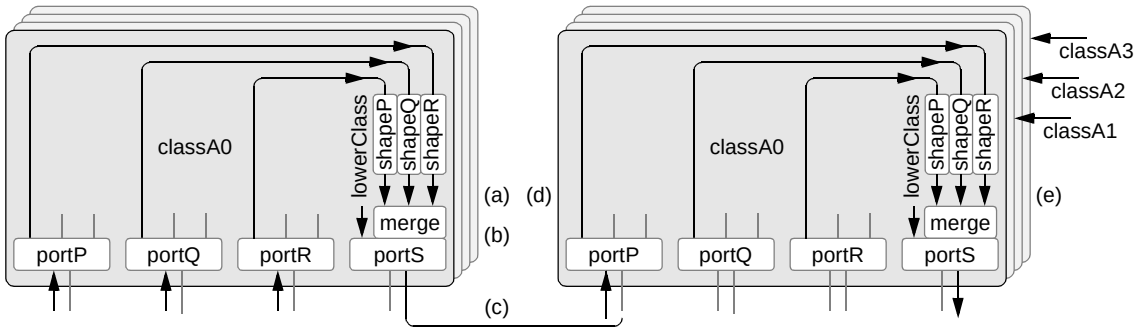

Figure 10.3—Reshaped bridge-traffic topology, with bunching control

In Figure 10.3, classA0 traffic flows between points (a, b, c, d, e), exhibits bunched and reshaped behaviors, as illustrated in Figure 10.4.

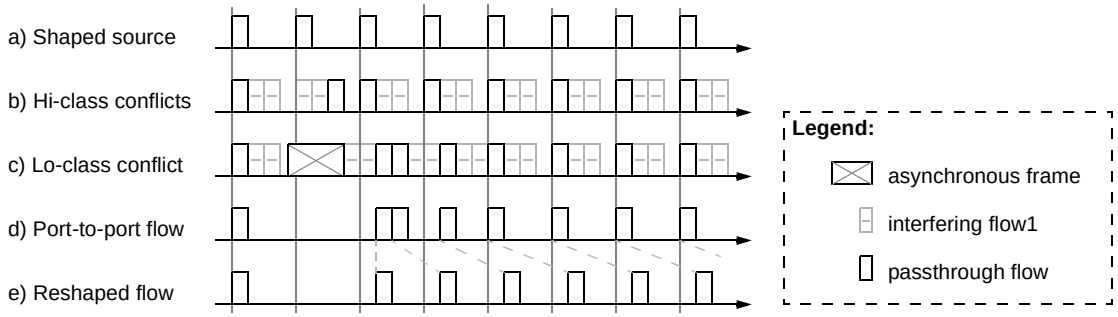

Figure 10.4—Reshaped bridge-traffic timing

The (a) through (e) time lines represent the flow of frames from within one talker-or-bridge into another bridge-or-listener, described as follows:

a) A properly shaped source stream is originally generated within a talker, or a port-to-port flow (consisting of multiple streams) within a bridge.

b) Forwarding of multiple sources to a shared transmission link can produce jitter, due to slight differences in frame-to-frame spacings.

c) Forwarding of multiple sources to a shared transmission link can produce additional jitter, when higher-class traffic waits for the completion of previously initiated lower-class transmissions.

d) Bunching becomes apparent in the port-to-port flow, representing the portion of the received (c) traffic that is forwarded to a specific transmitter port.

e) A shaper delays the forwarding of bunched frames, so that the port-to-port flow is properly shaped. Delays can be invoked by time stamping frames with an in-the-future transmission time.

The reshaped flow (e) retains the properly shaped properties of the preceding flow (a), while incurring a maximum delay *d* through the bridge. These properties ensure a linear maximum delay of $n \times d$, for streams that flow through N bridges.

10.1.3 Port-to-port reshaping, with bunching control

The complexity of managing traffic classes can be reduced by eliminating shapers, so that receiver inputs are merged before being shaped, as illustrated in Figure 10.5 (only the components associated with specific flows are illustrated). In this illustration, classA0 traffic flows between points (a, b, c, d, e), exhibits bunched and reshaped behaviors, as illustrated in Figure 10.4.

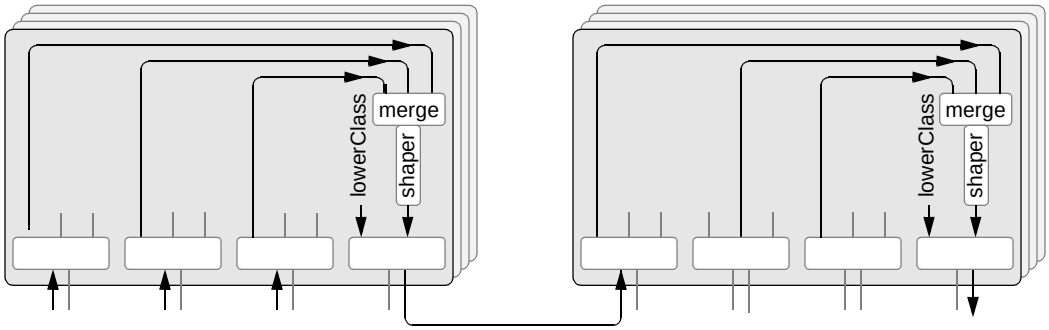

Figure 10.5—Reshaped bridge-traffic topology, without bunching control

This cost-reduced approach solves the bursting problem, without addressing the bunching problem. Within constrained topologies this can sometimes be sufficient to meet worst-case latency requirements.

10.1.4 Transmit ports

10.1.4.1 Transmit port structure

The transmit port is responsible for shaping classA traffic (to avoid bunching) and pacing classB/classC traffic (to avoid classC traffic starvation). Pacing and shaping algorithms assume functionally distinct queues within each transmit port, as illustrated in Figure 10.6.

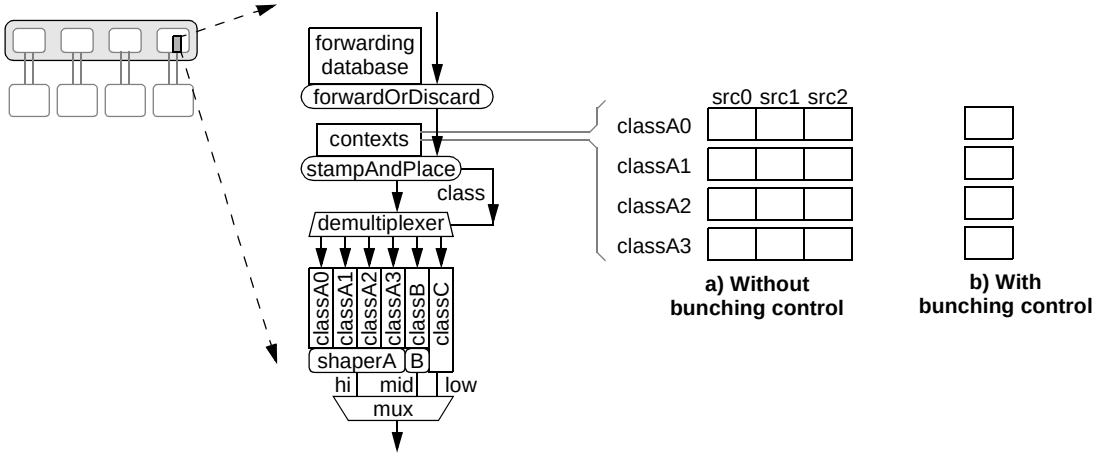

Figure 10.6—Transmit-queue structure

The intent of per-class shapers is to avoid priority inversions, wherein higher-class frames are delayed by the presence of concurrent lower-class traffic. Independent per-class shapers and queues allow enqueued higher-class and lower-class frames to be forwarded independently, thus avoiding priority inversions within queues.

The intent of per-source shapers is to avoid increasingly large cumulative bunching delays. The per-source reshaping eliminates bunches before merging, so that the pass-through bunching severity for 1-bridge and *n*-bridge flows are the same.

10.1.4.2 Enqueue reshaping contexts

The desired per-class latencies could not be guaranteed in the presence of classA traffic bunching. To avoid bunching, frames are shaped before being placed into classified transmit queues.

A shaper is responsible for attaching a time-stamp label to frames. With bursting control, a time-stamp shaper is logically associated with each source port and each classA traffic subclass (classA0, classA1, classA2, classA3). E.g, a four-port switch (which has three possible source ports) would have 12 time-stamp shapers on each transmit port. Without bursting control, a time-stamp shaper is logically associated with each classA traffic subclass (classA0, classA1, classA2, classA3). E.g, a N-port switch (which has N-1 possible source ports) would have 4 time-stamp shapers on each transmit port.

The purpose of a time-stamp shaper is to associate a time-stamp label with each queued frame. The time-stamp label represents a time in the future; the frame's transmission is deferred until the current time reaches the frame's time-stamp value. This facilitates the delayed forwarding of successive frames within each bunch, thus suppressing the bunching effects found on receive-link transmission.

The context for each time-stamp shaper is based on the frame's receive port and traffic class. While the context is considerably larger than that associated with strict per-port shapers, only one shaper (within each port) is ever active. Thus, context-switching per-port shaper instances represent a viable implementation technology.

10.1.4.3 Dequeue shaping and pacing

Transmit ports utilize a shaper and pacer, as illustrated as shaperA and B components within Figure 10.6. The purpose of these is to ensure forward progress of best-effort control traffic. In concept, this involves a two-step bandwidth partitioning mechanism:

a) The shaperA limits the cumulative classA and primary classB traffic to 75% of the link bandwidth. The intent is to ensure that 25% residual bandwidth remains available for lower-class traffic.

b) Pacer B partitions the residual 25% traffic equally between classB and classC traffic.
This ensures that classB traffic is never starved, in the presence of 75% classA traffic.
This ensures that classC traffic is not starved, in the presence of excess classB traffic.

10.1.5 Credit-based shapers and pacers

10.1.5.1 Credit-predictive shapers

The transmit shaper that schedules classA traffic (see Table 10.7, transition SHAPE-1) prioritizes frames based on their target transmission times. The shaper's credits are adjusted down or up, as illustrated in Figure 10.8. When frames are transmitted, the decrement value represents the size of a transmitted frame. Between transmissions, the credits increase based on the allowed per-class transmission rate.

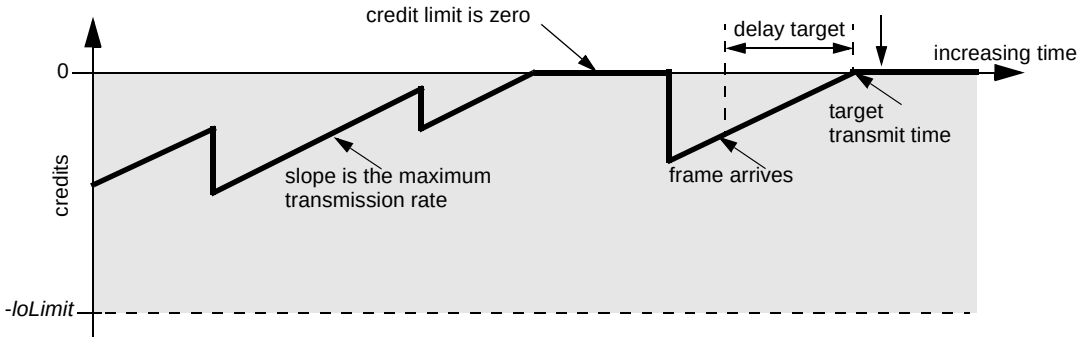

Figure 10.7—Credit-predictive shapers

Crossing above the zero threshold is disallowed, since this would encourage frame-transmission bunching. In normal operation, the credit value never goes below the *–loLimit* extreme.

When a frame arrives, the credits are evaluated to determine a target time for the frame transmission, based on credit losses from this and recent transmissions, divided by the allowed transmission rate. This provides a delay target time; the enqueued frame with the nearest delay-target time is selected for transmission.

In concept, the shaper consist of a token bucket. The number of credits in a token bucket is decremented by the size of each transmitted frame. The credits in the token bucket are incremented at the end of every credit update interval. A transmit time is targeted for when the debits (negative credits) are eliminated.

10.1.5.2 Credit-history shapers

The transmit shaper that limits levels of classA traffic (see Table 10.3, transitions BEST-1 to BEST-5) selectively enables transmissions based on an accumulated credits value. The shaper's credits are adjusted down or up, as illustrated in Figure 10.8. When frames are transmitted, the decrement value represents the size of a transmitted frame. Between transmissions, the credits increase based on the maximum allowed transmission rate.

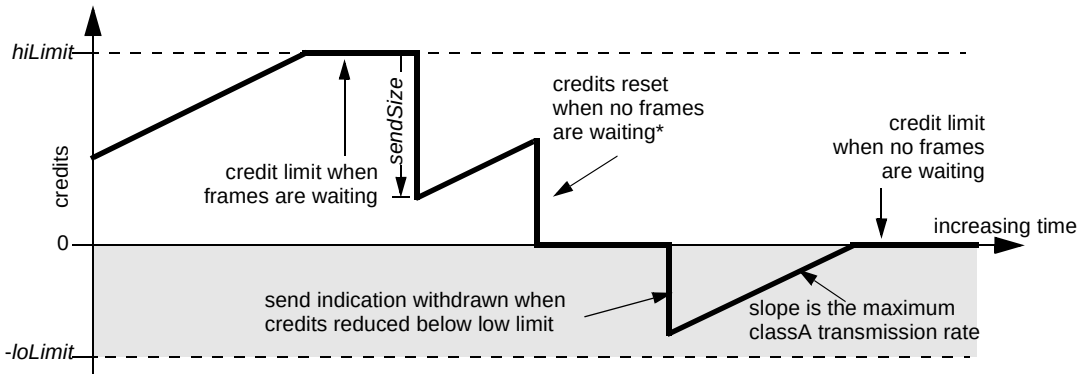

Figure 10.8—Credit-based shapers

Crossing below the zero threshold generates a rate-limiting 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 shaper consist of a token bucket. The number of credits in a token bucket is decremented by the size of each transmitted frame. The credits in the token bucket are continually increasing over time. A frame is only transmitted when the credits are positive.

10.1.5.3 Credit-based pacers

Although multiple pacers are specified within this working paper, the behavior of most pacers can be characterized by a common algorithm and instance-specific parameters (as done within RPR[B5]). The pacer's credits are adjusted down or up, as illustrated in Figure 10.9. The decrement and increment values typically represent sizes of debit and credit frames in each update interval, respectively.

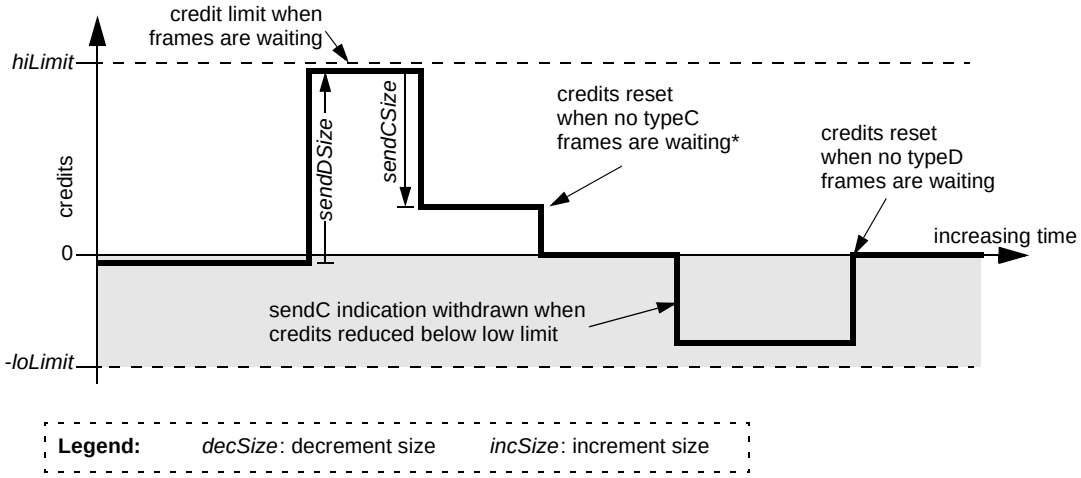

Figure 10.9—Pacer credit adjustments over time

Crossing below the zero threshold generates a rate-limiting 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 pacer consists of a token bucket. The credits in the token bucket are incremented by the size of each transmitted debit-frame. The number of credits in a token bucket is decremented by the size of each transmitted credit-frame. A credit-frame is only transmitted when the credits are positive; a debit-frame is only transmitted when the credits are negative.

10.2 Terminology and variables

10.2.1 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 transmit port's internal queue, where received frames are placed.
QP_TX_A0—The first of the output port's classA buffers.
QP_TX_A1—The second of the output port's classA buffers.
QP_TX_A2—The third of the output port's classA buffers.
QP_TX_A3—The fourth of the output port's classA buffers.
QP_TX_BP—The output port's classB queue.
QP_TX_CP—The output port's classC queue.
QP_TX_LINK—The output port's transmit-PHY queue.

10.2.2 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.
framed
The contents of a received frame, with supplemental information, as follows:
frame—The contents of a frame.
sourcePort—The source port that received the frame.
txTime—A time-stamp value representing the intended (bunching delayed) transmission time.

10.2.3 Common state machine routines

Max(*value1*, *value2*)
Returns the numerically larger of two values.

10.2.4 Variables and routines defined in other clauses

This clause references the following variables and routines defined in Clause 7:

currentTime
See 7.2.2.
Dequeue(*queue*)
Enqueue(*queue*, *frame*)
Min(*value1*, *value2*)
See 7.2.3.

10.3 Pacing state machines

10.3.1 TransmitRx state machine

The TransmitRx state machine is responsible for enqueuing traffic (received on other ports and broadcast to all possible transmitter ports) for possible forwarding. An intent is to transfer each to the appropriate output queue.

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

10.3.1.1 TransmitRx state machine definitions

queue values
Enumerated values used to specify shared queue structures.
QP_TX_A0, QP_TX_A1, QP_TX_A2, QP_TX_A3
QP_TX_BP, QP_TX_CP
QP_TX_PUSH
See 10.2.1.

10.3.1.2 TransmitRx state machine variables

credits
A value that corresponds to accumulated credits since the last frame transmission.
class
A value that represents the frame's priority class.
currentTime
See 10.2.4.
delay
A value that represents the time delay assigned by the frame's shaper.
framed
See 10.2.2.
sPtr
Represents a pointer to shaper values.

10.3.1.3 TransmitRx state machine routines

ContextCheck(*sourcePort, class*)
Returns a pointer to the associated pacer context, with the following fields:
credit—The cumulative credit from past pacer activities.
lastTime—The last time the pacer was invoked.
loLimit—The low limit for shaper credits.
rate—The highest allowed rate of the paced traffic, in bytes-per-second.
Dequeue(*queue*)
See 10.2.4.
Enqueue(*queue, frame*)
Places the *frame* at the tail of the specified *queue* within the assumed port.
ForwardClass(*framed*)
The forwarding database is checked. If forwarding is enabled, the priority class is returned.
Otherwise, a NULL class value is returned. The following enumerated values are returned:
CLASS_A0—The associated multicast frame is forwarded as classA traffic.
CLASS_A1—The associated multicast frame is forwarded as classA traffic.
CLASS_A2—The associated multicast frame is forwarded as classA traffic.
CLASS_A3—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.
Max(*value1, value2*)
See 10.2.3.
Min(*value1, value2*)
See 10.2.4.

10.3.1.4 TransmitRx state table

The TransmitRx state machine is specified in Table 10.2. 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 10.2—TransmitRx state table

Current		Row	Next	
state	condition		action	state
START	(framed = Dequeue(QP_TX_PUSH))!=NULL	1	—	FIRST
	—	2	—	START
FIRST	(class = ForwardClass(framed)) != NULL	1	sPtr = ContextCheck(framed.sourcePort, class);	NEXT
	—	2	—	START
NEXT	class == CLASS_A0	1	queue = QP_TX_A0;	SHAPE
	class == CLASS_A1	2	queue = QP_TX_A1;	
	class == CLASS_A2	3	queue = QP_TX_A2;	
	class == CLASS_A3	4	queue = QP_TX_A3;	
	class == CLASS_B	5	queue = QP_TX_BP;	FINAL
	—	6	queue = QP_TX_CP;	
SHAPE	—	1	credits = sPtr–>rate * (currentTime – sPtr–>lastTime); sPtr–>lastTime = currentTime; sPtr–>credit = Max(0, Min(sPtr–>loLimit, sPtr–>credit + credits – Size(frame))); framed.txTime = currentTime – sPtr–>credit / sptr–>rate;	FINAL
FINAL	—	1	Enqueue(queue, frame);	START

START-1: If a frame has arrived, process that frame.
START-2: Otherwise, wait for the next frame to arrive.

FIRST-1: When forwarded frames, the shaper context is based on the source port and class.
FIRST-2: The non-forwarded frames are discarded.

NEXT-1: The classA0 frames are forwarded to the appropriate time-sensitive classA0 queue.
NEXT-2: The classA1 frames are forwarded to the appropriate time-sensitive classA1 queue.
NEXT-3: The classA2 frames are forwarded to the appropriate time-sensitive classA2 queue.
NEXT-4: The classA3 frames are forwarded to the appropriate time-sensitive classA3 queue.
NEXT-5: The classB frames are forwarded to the appropriate time-sensitive classB queue.
NEXT-6: The classC frames are forwarded to the appropriate time-sensitive classC queue.

SHAPE-1: ClassA frames are time stamped by shapers.
Shaper pacer parameters for each distinct {*class, source*} pair constrains bunching within each class.
Shaper parameters are decremented on transmissions and incremented by *credits*, where *credits* corresponds to the credits accumulated since the last applicable frame transmission (a specified in 10.1.5.2).
The last-updated time is then updated, to account for the incremented credits.
High and low limits are applied to the updated credits, thus avoiding burst-related positive credits.
Negative credits correspond to delayed transmission times, affiliated with output-port-queue frames.

FINAL-1: The received frames are placed into the appropriate queue.

10.3.2 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.

10.3.2.1 TransmitTx state machine definitions

BPS
: The nominal link transmission rate, in bytes per second.

MTU
: The maximum frame size, in bytes.

queue values
: Enumerated values used to specify shared queue structures.
  QP_TX_A0, QP_TX_A1, QP_TX_A2, QP_TX_A3
  QP_TX_BP, QP_TX_CP
  QP_TX_LINK
  See 10.2.1.

TICK
: The amount of time between shaper updates.
  Range: [1 bytes transmit time, 16-bit transmit time]
  Default: 1 byte transmit time

10.3.2.2 TransmitTx state machine variables

best
: A value that represents the weight and identify of the next-best classA queue.
  goodness—The smallest *weight*×*wait* value associated with alternate classA transmissions.
  queue—The queue associated with the best futuristic encapsulated frame.

countA
: A speculative value of creditA, used only when the frame is qualitified for transmission.

countB
: A speculative value of creditB, used only when the frame is qualitified for transmission.

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

creditB
: A shaper credit value whose positive and negative values enable secondary classB and classC transmissions respectively.

currentTime
: See 10.2.4.

frame
: The contents of a to-be-transmitted frame.

framed
See 10.2.2.
hiLimitA
A value that limits the cumulative *creditA* credits.
Value: MTU.
hiLimitB
A value that limits the cumulative *creditB* credits.
Value: MTU.
limit
A value that limits the amount of transmitted primary classA/classB bandwidth.
loLimitA
A value that limits the cumulative *creditA* debits.
Value: MTU.
loLimitB
A value that limits the cumulative *creditB* debits.
Value: MTU.
tickTime
A value that defines when the time-tick interval ends.

10.3.2.3 TransmitTx state machine routines

Dequeue(*queue*)
See 10.2.4.
Unqueue(*queue*, *weight*, *&best*, *currentTime*)
Dequeues and returns the most overdue frame from the specified *queue*, excluding those frames whose scheduled transmission time is after the specified *currentTime* value.
framed—The oldest of the overdue frame.
NULL—No frame available.
In the presence of only futuristic frames, a *test*= *weight*×(*txTime-currentTime*) value is computed.
If *best.queue* is NULL or *test*<*best.goodness*, the *best.queue* and *best.goodness* components are updated to reflect the best alternate classA transmission queue.
Enqueue(*queue*, *frame*)
See 10.2.4.
Size(*frame*)
Returns the size of the specified frame.
StaleFrame(*frame*, *queue*)
Indicates whether the specified frame is stale and discardable, as specified by Equation 10.1.
0—The specified frame is not stale.
1—(Otherwise.)

```
// The value of "internal" depends on the class, as specified in Table 10.1.
(index = (QP_TX_A0 - queue),
(currentTime - framed.txTime) > (2 * (MTU + interval[index] * BPS)))
```
(10.1)

10.3.2.4 TransmitTx state table

The TransmitTx state machine is specified in Table 10.3. 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 10.3—TransmitTx state table

Current		Row	Next	
state	**condition**		**action**	**state**
START	(currentTime – tickTime) >= TICK;	1	creditA = Min(hiLimitA, creditA + 0.75 * TICK * BPS); tickTime = currentTime ;	START
	TransmissionInProgress()	2	—	
	creditA < 0	3	—	FAIR
	—	4	best.queue = NULL;	BEST
BEST	(framed = Unqueue(queue= QP_TX_A0, 32, &best, currentTime)) != NULL	1	countA = Min(loLimitA, creditA – Size(framed));	NEAR
	(framed = Unqueue(queue= QP_TX_A1, 16, &best, currentTime)) != NULL	2		
	(framed = Unqueue(queue= QP_TX_A2, 8, &best, currentTime)) != NULL	3		
	(framed = Unqueue(queue= QP_TX_A3, 4, &best, currentTime)) != NULL	4		
	best.queue != NULL && (framed = Dequeue(queue= best.queue)) != NULL	5		
	(framed = Dequeue(QP_TX_BP)) != NULL	6	creditA = Min(loLimitA, creditA – Size(framed));	FINAL
	—	7	creditA = 0;	START
FAIR	creditB >= 0 && (framed = Dequeue(QP_TX_BP)) != NULL	1	creditB = creditB – Size(framed);	FINAL
	creditB <= 0 && (framed = Dequeue(QP_TX_CP)) != NULL	2	creditB = creditB + Size(framed);	
	(framed = Dequeue(QP_TX_BP)) != NULL	3	creditB = 0;	
	(framed = Dequeue(QP_TX_CP)) != NULL	4		
	—	5	creditB = 0;	START
NEAR	StaleFrame(framed, queue)	1	—	START
	—	2	creditA = countA;	FINAL
FINAL	—	1	Enqueue(QP_TX_LINK, framed.frame);	START

START-1: Update the classA credits after each tick interval.
START-2: Wait for the queue to be emptied, so that something can be transmitted.

START-3: In the absence of classA credits, fairly transmit enqueued classB and classC frames.
START-4: Fairly service classB/classC when the classA/classB transmissions are disallowed.

BEST-1: If enabled and available, a classA0 frame is transmitted.
BEST-2: If enabled and available, a classA1frame is transmitted.
BEST-3: If enabled and available, a classA2 frame is transmitted.
BEST-4: If enabled and available, a classA3 frame is transmitted.
BEST-5: If available, a scheduled-for-the-future classA frame is transmitted.
BEST-6: If enabled and available, a classB frame is transmitted.
BEST-7: Since nothing is ready to be sent, the classA credits are cleared.

FAIR-1: If enabled and available, a classB frame is transmitted.
The *creditB* values is decremented by the transmitted frame size, to avoid classC starvation.
FAIR-2: If enabled and available, a classC frame is transmitted.
The *creditB* values is incremented by the transmitted frame size, to avoid classB starvation.
FAIR-3: If available, a classB frame is transmitted.
FAIR-4: If available, a classC frame is transmitted.
FAIR-5: Otherwise, no frame is transmitted.

NEAR-1: Stale frames, whose delivery times cannot be guaranteed, are discarded.
NEAR-2: Non-stale frames are not discarded.

FINAL-1: The next frame is transmitted and credits are updated accordingly.