dvjConservative2003Nov08.fm

November 10, 2003 12:57 pm

Refined fairness

This contribution provides replacement text for Clause 9 of P802.17 RPR D2.6. Several types of changes are proposed:

a) Less extremes. The minimum and maximum priority levels are 16 and 255 respectively.

b) Normalized. This specification specifies behavior time-constants, not design-specific numbers.

c) Conservative simplified.

   1) No hysteresis.
   2) No round-trip delay dependency.
   3) Initial target is based on history, rather than complex weighted transit-traffic monitors.

d) Scope. A *scope* parameter allows delayed reaction to ill-formed congestion indications.

The following TBDs are being considered.

a) For continuity, this draft supports rate-limits through shapers or rate comparisons. For simplicity, perhaps only the (self-calibrating) shapers should be specified.

The following outstanding questions are being investigated.

a) Why do we generate *lpAddRateCongested*? Its initialized and generated, but apparently not used.
Confirmation: The *addRateCongested* is needed for rate-based throttles; the editors included the low-pass filtered version of this value for apparent completeness.

b) Are the hops to congestion properly measured?
Confirmation: An increment is needed in page 289, line 28:
==> hopsToCongestion = ringInfo.totalHopsTx[myRI] + 1;
Confirmation: An increment is needed in page 289, line 31:
==> hopsToCongestion = MAX_STATIONS - rcvdTtl + 1;

c) Most of the code seems to generate the next fairness message based on what was last received. Does this work on startup or on a steered edge, where no such information has been received?
Confirmation: Since nothing is being sent through this station, it will become a congestion head. Thus, the values of the transmitted fairness frame appear irrelevant.
Speculation: Woops! What happens if the last fairness frame were from the other ri?

d) In D2.7, Table-Row 9.6-11, isn't the following qualification condition redundant:
   nrXmitRate < unreservedRate[myRI] &&
Speculation: This could redundant, but possibly depends on the downstream shaper details.

e) In D2.7, Table-Row 9.9-10, there is a statement:
   allowedRate = Min(unreservedRate[myRI], localFairRate);
Speculation: This requirement could be fulfilled by precomputing the following:
   maxAllowedRate2 = Min(maxAllowedRate, unreservedRate[myRI]);
and then using *maxAllowedRate2* where *maxAllowedRate* is now used?

9.1.1 Aggressive behaviors

The computation of *localFairRate* value is based on congestion conditions, as illustrated Figure 9.12. When uncongested, the *localFairRate* value is set to FULL_RATE and *allowedRate* approaches an *lpAddRate*/*weight* traffic limit. When uncongested, *localFairRate* and *allowedRate* are immediately set to the *lpAddRate*/*weight*.

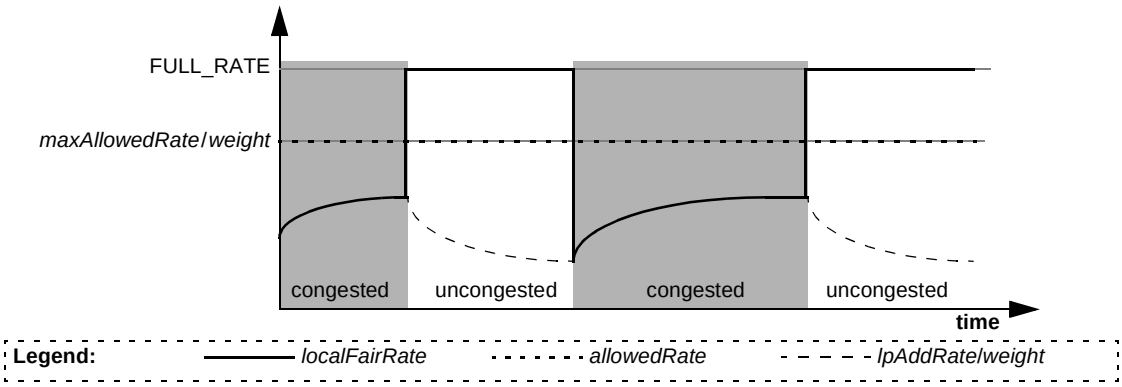

Figure 9.12—Aggressive tracking of *localFairRate* computations

9.1.2 Conservative behaviors

The computation of *localFairRate* value is based on *loAddRate/weight* congestion-condition estimates, computed as illustrated Figure 9.13. When congested, the *localFairRate* value approaches an *lpAddRate*/*weight* traffic rate; when uncongested, *localFairRate* approaches the FULL_RATE value.

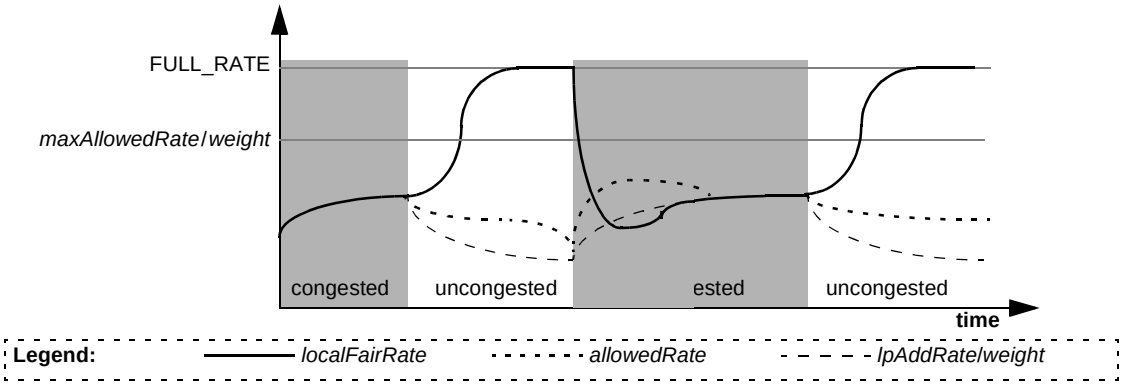

Figure 9.13—Tracking of *localFairRate* computations

The *allowedRate* tracks the *localFairRate* during congested intervals, while remaining constrained to a *maxAllowedRate*/*weight* value during uncongested intervals.

9.1.3 Congestion domains

A congestion domain ends when an upstream station A senses no need to inhibit contention-point traffic at downstream station B, as illustrated Figure 9.14.

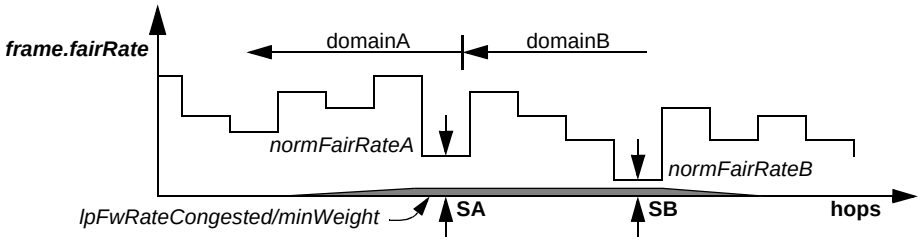

Figure 9.14—Fairness domain boundaries

Station SA is aware of downstream congestion point at station SB, based on its reception of the *normFairRateB=fairFrame.fairRate* value. Station SA is also aware of *lpFwRateCongested/minWeight*, a conservative estimate of the upstream transmission rates.

If the upstream transmission rates are less than their restricted-by-*normFairRateB* values, station SA assumes these stations flows would also be unaffected by a larger *normFairRateA* value. Thus, station SA can communicate a *fairFrame.fairRate=normFairRateA* to its upstream neighbors, without impacting traffic through the more congested downstream station SB.

9.2 …

9.2.1 Common state machine definitions, variables, and routines

9.2.1.1 Common state machine definitions

The definitions listed in this section are referenced by multiple state machines.

FULL_RATE
: A internal value corresponding to a full-scale line-rate value.
  Value: $3FFFFFFF_{16}$

SCALE
: The scalar that converts the advertised and internal full-scale values.
  Value: (1 << 14)

9.2.1.2 Common state machine variables

addRate
: The rate that fairness-eligible traffic is added. (Line-rate corresponds to FULL_RATE.)

addRateCongested
: The rate that fairness eligible traffic intended for destinations downstream of the congestion point, is added to the ringlet. (Line-rate corresponds to FULL_RATE).

addRateCongestedOk
: Indicates whether fairness eligible traffic bound for a destination beyond the congestion point is allowed to be added to the ringlet.
  TRUE—Traffic can be added.
  FALSE—(Otherwise.)

addRateOk
: Indicates whether fairness eligible traffic is allowed to be added to the ringlet.

TRUE—Traffic can be added.
FALSE—(Otherwise.)

admissionMethod
An enumerated value used to specify the traffic admission method:
RATE_BASED—Admission is based on rate policing.
SHAPER_BASED—Admission is based on rate shaping.

allowedRate
The rate at which fairness eligible traffic can be added to the ring. (Line-rate corresponds to FULL_RATE.)

allowedRateCongested
The rate at which a station is allowed to add fairness eligible traffic intended for destinations downstream of the congestion point. (Line-rate corresponds to FULL_RATE.)

fwRate
The rate that fairness-eligible traffic transits the station. (Line-rate corresponds to FULL_RATE.)

fwRateCongested
The rate at which fairness-eligible traffic intended for destinations downstream of the congestion point transits the station. (Line-rate corresponds to FULL_RATE.)

localCongested
Indicates whether or not the station is locally congested.
TRUE—The station is locally congested.
FALSE—(Otherwise.)

lpFwRate
A smoothed (low-pass filtered) version of *fwRate*.

lpFwRateCongested
A smoothed (low-pass filtered) version of the *fwRateCongested*.

localFairRate
The advertised local-station transmission rate by which a station limits upstream fairness eligible traffic. (Line-rate corresponds to FULL_RATE.)

lpNrRate
A smoothed (low-pass filtered) version of the *nrRate*.

maxAllowedRate
The maximum permitted value of the *allowedRate*. (Line-rate corresponds to FULL_RATE.)

maxAllowedRate2
The minimum of *maxAllowedRate* and *unreservedRate*[*myRI*] allowed rates.

normFairRate
A scaled version of *localFairRate* for comparing to the fairness frame's *fairRate* value. (Line-rate corresponds to FULL_RATE/weight.)

nrRate
The rate that non-reserved traffic is transmitted. (Line-rate corresponds to FULL_RATE).

receivedHops
The distance, in hops, between the local station and the head of the congestion domain. The value of MAX_STATIONS indicates the local station does not lie within a congestion domain.

receivedRate
The effective congestion rate derived from the last received fairness frame.

receivedScope
The scope of congestion-report applicability. This is cleared to zero whenever fairness frames return to a less-congested self, thereby indicating the suspect nature of its reported values.

rxFrame
The contents of a received RPR frame.

stqHighThreshold
A level of STQ occupancy at or above which fairness eligible add frames are no longer admitted. Defined only for a dual-queue implementation.
Range: [3 * *mtuSize*, *stqFullThreshold* - *mtuSize*]
Default: 0.25 * *stqFullThreshold*

stqLowThreshold

A level of STQ occupancy at or above which congestion on the outbound link is imminent. Defined only for dual-queue implementations.

Range: [*mtuSize*, *stqMedThreshold* - *mtuSize*]

Default: 0.5 * *stqHighThreshold*

9.2.1.3 Common state machine routines

Hops(*frame*)

Indicates the distance from the fairness frame's source station, as defined by Equation 9.1.

```
(MAX_STATIONS – rxFrame.ttl + 1)
```
(9.1)

Mul(*value1*, *value2*)

An unsigned multiplication by a 30-bit binary fraction, defined by Equation 9.2.

```
(((uInt64_t)value1 * value2) / SCALE)
```
(9.2)

Div(*value1*, *value2*)

An unsigned division by a 30-bit binary fraction, defined by Equation 9.3.

```
(((uInt64_t)value1 * SCALE) / value2)
```
(9.3)

MultiChokeInd( )

Transfer to the MAC client via an MA_CONTROL.indication having an opcode of MULTI_CHOKE_IND (see Table 5.7).

ScaleDown(*value*)

An operation that converts between internal-scaled and advertised-scaled values.

```
((value - (value >> 16)) / SCALE)
```
(9.4)

ScaleDown(*value*)

An operation that converts the internal-scaled and advertised-scaled values.

```
((value * SCALE) + ((value * SCALE) >> 10))
```
(9.5)

9.2.2 Literals, variables, and routines defined in other clauses

This clause references the following variables and routines defined in Clause 6.

classBAccessTime
classCAccessTime
currentTime
EntryInQueue( )
Min(*value1*, *value2*)
myDualQueueStation
myEdgeState
myMacAddress
myProtectMethod
myRI
Other(*ri*)
passAddFe
passAddFeOrStq
passAddFeCongested
sendD
stqDepth

This clause references the following literals and variables defined in Clause 10:

conservativeMode
MAX_STATIONS
myTopoInfo.unreservedRate[*ri*]
ringInfo.multichokeUser[*ri*]
ringInfo.totalHopsTx[*ri*]
TransmitFrame(*frame*)

9.2.3 FairessRating state machine

9.2.3.1 FairnessRating state machine definitions

The definitions listed in this section are referenced by multiple state machines.

BYTES_PER_AGING
The maximum number of bytes included in an aging interval.
Value: TBD

ONE_BYTE
An incremental rate corresponding to one fairness-eligible byte, as defined by Equation 9.6.

```
(FAIR_RATE1 / (BYTES_PER_AGING * (admissionMethod==RATE_BASED ? AGING_COEF : 1)))
```
(9.6)

9.2.3.2 FairnessRating state machine variables

addRate
addRateCongested
admissionMethod
See 9.2.1.2.
advertisementInterval
The length of an advertisement interval.
advertisementTime
The time at the start of the current advertisement interval.
agingInterval
The length of an aging interval.
agingTime
The time at the start of the current aging interval.
byteAdded
Set to 1 when an added byte is transmitted.
byteTransited
Set to 1 when an transited byte is transmitted.
currentTime
See 9.2.2.
fwRate
fwRateCongested
See 9.2.1.2.
myDualQueueStation
See 9.2.2.
nrRate
receivedHops
rxFrame
See 9.2.1.2.
thisFrame
The contents of the current frame being transmitted.

9.2.3.3 FairnessRating functions

FairnessAging( )

Invokes the actions of the state machine specified in 9.2.4.

FairnessEligible(*frame*)

Indicates when the frame is fairless eligible, as specified in Equation 9.7.

```
((frame.sc == CLASS_B && frame.fe == 1) || frame.sc == CLASS_C)        (9.7)
```

FairnessReceive( )

Invokes the actions of the state machine specified in 9.2.8.

FairnessTransmit( )

Invokes the actions of the state machine specified in 9.2.9.

ReceiveScfFrame( )

Provides a received single-choke fairness frame, if one has been received since last called.

value—The currently the last received single-choke fairness frame.

(null)—No single-choke fairness frame is available.

9.2.3.4 FairnessRating state table

The FairnessRating state table counts the bytes in each aging interval, as specified in Table 9.1.

Table 9.1—FairnessRating state table

Current state		Row	Next state	
state	condition		action	state
START	byteAdded	1	byteAdded= FALSE;	ADDS
	byteTransited	2	byteTransited= FALSE;	FORW
	(currentTime - agingTime) >= agingInterval	3	FairnessAging(); agingTime = currentTime;	START
	(rxFrame= ReceiveScfFrame()) != NULL	4	FairnessReceive();	
	(currentTime - advertisementTime) >= advertisementInterval	5	FairnessTransmit(); advertisementTime = currentTime;	
ADDS	FairEligible(thisFrame) == FALSE	6	—	PLUS
	admissionMethod == RATE_BASED && frame.ttl > receivedHops	7	addRate += ONE_BYTE; addRateCongested += ONE_BYTE;	
	—	8	addRate += ONE_BYTE;	
FORW	FairEligible(thisFrame) == FALSE	9	—	PLUS
	frame.ttl > receivedHops	10	fwRateCongested += ONE_BYTE; fwRate += ONE_BYTE;	
	—	11	fwRate += ONE_BYTE;	
PLUS	myDualQueueStation == FALSE && frame.sc != CLASS_A0	12	nrRate += ONE_BYTE;	START
	—	13	—	

Row 9.1-1: This station's fairness-eligible added bytes are counted.
Row 9.1-2: This station's fairness-eligible transited bytes are counted.
Row 9.1-3: Fairness aging occurs once every aging interval.
Row 9.1-4: Fairness frames are transmitted periodically.
Row 9.1-5: Fairness frames are processed when received.

Row 9.1-6: The *addRate* counters only apply to fairness-eligible traffic.
Row 9.1-7: (Optional: Necessary to support the rate-based admission method.)
Rate-based admission methods calculate the *addRateCongested* value.
Row 9.1-8: When transmitting before congestion, update only *addRate*.

Row 9.1-9: The *fwRate* counters only apply to fairness-eligible traffic.
Row 9.1-10: When transiting beyond congestion, update both *fwRate* and *fwRateCongested*.
Row 9.1-11: When transiting before congestion, update only *fwRate*.

Row 9.1-12: Row 9.1-13: (Optional: Necessary to support single-queue designs.)
The *nrRate* value accounts for non-reserved byte transmissions.

9.2.4 FairnessAging state machine

9.2.4.1 FairnessAging state machine definitions

NORM
: A low-pass filter with a ringlet-delay dependent time constant. This value is nominally set to:
(2**agingInterval*)/*ringletLoopDelay*
Where *ringletLoopDelay* is the nominal classA-traffic ringlet circulation time.

RAMP_NORM
: A low-pass filter ramp-up time constant. This value is nominally set to NORM.

RAMP_SLOW
: A low-pass filter ramp-up time constant. This value is nominally set to NORM/4.

9.2.4.2 FairnessAging state machine variables

addRate

addRateCongested

admissionMethod
: See 9.2.1.2.

ageCoef
: The coefficient used by the aging procedure to specify an effective filter time constant.
Allowed: {1, 2, 4, 8, 16}
Default: 4

allowedRate

allowedRateCongested
: See 9.2.1.2.

baseAllowedRate
: The value of *allowedRate*/*weight*.

baseAllowedRateCongested
: The value of *allowedRateCongested*/*weight*.

fwRate

fwRateCongested

localFairRate
: See 9.2.1.2.

lpAddRate
: A smoothed (low-pass filtered) version of *fwRate*.

lpFwRate

lpFwRateCongested

maxAllowedRate
: See 9.2.1.2.

myDualQueueStation
: See 9.2.2.

receivedRate

receivedScope
: See 9.2.1.2.

target
: A temporary variable that retains the value of intermediate calculations.

weight
: The fairness weighting associated with this fairness instance.

9.2.4.3 FairnessAging state machine routines

FairnessChecking( )
Invokes the actions of the state machine specified in 9.2.5.
Min(*value1*, *value2*)
See 9.2.1.3.
Min3(*value1*, *value2*, *value3*)
Returns the numerically smaller of the three argument values.
Mul(*value1*, *value2*)
See 9.2.1.3.

9.2.4.4 FairnessAging state table

The FairnessAging state table, as specified in Table 9.2.

Table 9.2—FairnessAging state table

Current state		Row	Next state	
state	condition		action	state

START	admissionMethod == RATE_BASED	1	addRate -= addRate/ageCoef;
			addRateCongested -= addRateCongested/ageCoef;
			fwRate -= fwRate/ageCoef;
			fwRateCongested -= fwRateCongested/ageCoef;
				FIRST

	—	2	—	

FIRST	—	3	lpAddRate += Mul(addRate - lpAddRate, NORM);
			lpFwRate += Mul(fwRate - lpFwRate, NORM);
			lpFwRateCongested += Mul(fwRateCongested - lpFwRateCongested, NORM);
				NEXT

NEXT	!myDualQueueStation	4	lpNrRate += Mul(nrRate - lpNrRate, NORM);	PAST
	conservativeMode	5		
	—	6	—	

PAST	admissionMethod != RATE_BASED	7	addRate = 0;
			fwRate = 0;
			fwRateCongested = 0;
				EXEC

	—	8	—	
EXEC	—	9	FairnessChecking();	NEAR
NEAR	receivedRate > allowedRateCongested	10	target= Min3(receivedRate, maxAllowedRate2/weight, 2*allowedRateCongested); baseAllowedRateCongested += Mul(target - baseAllowedRateCongested, (receivedScope != 0 ? RAMP_NORM : RAMP_SLOW));	FINAL
	—	11	baseAllowedRateCongested = receivedRate;	
FINAL	—	12	allowedRateCongested = baseAllowedRateCongested * weight; SingleChokeIndication( );	RETN

Row 9.2-1: For rate-based admission methods, add rates are filtered based on *ageCoef*.
Row 9.2-2: For shaper-based admission methods, add rates need not be filtered.

Row 9.2-3: Low-pass filters are applied at the end of each aging interval.

Row 9.2-4: (Optional: Necessary for single-queue designs.)
Row 9.2-5: (Optional: Necessary for conservativeMode designs.)
The nonreserved rate is low-pass filtered to determine levels of consumed bandwidth.
Row 9.2-6: Nonreserved traffic is excluded from this computation.

Row 9.2-7: For shaper-based admission methods, add rates are cleared before the next aging interval starts.
Row 9.2-8: For rate-based admission methods, add rates are not cleared.

Row 9.2-9: Conservative or aggressive fairness can be selected.

Row 9.2-10: When congestion is relieved, *allowedRateCongested* ramps towards its target value.
Row 9.2-11: When congestion increases, *allowedRateCongested* inherits the observed value.
Row 9.2-12: A SingleChokeIndication is sent to the client.

9.2.5 FairnessChecking state machine

9.2.5.1 FairnessChecking state machine definitions

FAST
: A low-pass filter with a fast time constant. This value is nominally set to 4*NORM.

ONE
: A multiplicative scalar value with a full-scale value of 40000000_{16}.

9.2.5.2 FairnessChecking state machine variables

allowed
: A temporary variable that retains the value of intermediate calculations.

allowedRate
: See 9.2.1.2.

conservativeMode
: See 9.2.2.

localCongested

localFairRate
: See 9.2.1.2.

lpAddRate
: A smoothed (low-pass filtered) version of *fwRate*.

lpNrRate

maxAllowedRate2
: See 9.2.1.2.

myTopoInfo.unreservedRate[*ri*]
: See 9.2.2.

stqDepth
: See 9.2.2.

stqLowThreshold

stqLowThreshold
: See 9.2.1.2.

target
: A temporary variable that retains the value of intermediate calculations.

weight
: The fairness weighting associated with this fairness instance.

9.2.5.3 FairnessChecking state machine routines

Div(*value1*, *value2*)
: See 9.2.1.3.

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

Min(*value1*, *value2*)
: See 9.2.1.3.

Min3(*value1*, *value2*, *value3*)
: Returns the numerically smaller of the three argument values.

Mul(*value1*, *value2*)
: See 9.2.1.3.

9.2.5.4 FairnessChecking state table

The FairnessChecking state table provides aggressive or conservative rate-limiting feedback, as specified in Table 9.3.

Table 9.3—FairnessChecking state table

Current state		Row	Next state	
state	condition		action	state
START	conservativeMode	1	—	CONS
	—	2	—	AGGR
AGGR	localCongested	3	localFairRate= lpAddRate/weight;	AGGR2
	—	4	localFairRate= FULL_RATE;	
AGGR2	—	5	allowedRate = maxAllowedRate2;	RETN
CONS	localCongested	6	localFairRate += Mul(lpAddRate/weight - localFairRate, FAST);	CONS2
	—	7	target = Min(FULL_RATE, Max(2*localFairRate, maxAllocatedRate2/weight)); localFairRate += Mul(target - localFairRate, FAST);	
CONS2	—	8	allowed = Min(FULL_RATE, lpAddRate + Max(0, Min(myTopoInfo.unreservedRate[myRI] - lpNrRate, maxAllowedRate2 - lpAddRate)) / 2);	CONS3
CONS3	myDualQueueStation	9	allowedRate = Mul(allow, Max(0, Min(ONE, Div(stqHighThreshold - stqDepth, stqHighThreshold - stqLowThreshold))));	RETURN
	—	10	allowedRate = allowed;	

Row 9.3-1: The aggressive mode may be supported.
Row 9.3-2: The conservative mode may be supported.

(Optional: The AGGR and AGGR2 states are necessary to support aggressive fairness.)
Row 9.3-3: For aggressive/congested, *localFairRate* is based on the weighted averaged add rate.
Row 9.3-4: For aggressive/uncongested, *localFairRate* is set to the largest (unconstrained) add rate value.
Row 9.3-5: For all aggressive conditions, transmissions are bounded only by the *maxAllowedRate* limit.

(Optional: The {CONS, CONS2, CONS3} states are necessary to support conservative fairness.)
Row 9.3-6: When congested, *myFairRate* tracks the *addRate* measurement.
Row 9.3-7: When not congested, *myFairRate* tracks the FULL_RATE target.

Row 9.3-8: The *allowed* rate is no larger than *lpAddRate* by half of the smallest: a) unused non-reserved bandwidth or b) remaining *maxAllowedRate2*-constrained bandwidth.

Row 9.3-9: (Optional: This row is necessary to support dual-queue station conservative fairness.) Reduce *allowedRate* as the *stqDepth* transitions from *stqLowThreshold* to *stqHighThreshold*.
Row 9.3-10: On a single-queue stations, *allowedRate* is uninfluenced by *stqDepth* depths.

9.2.6 CongestionSense state machine

The CongestionSense state machine updates the *localCongested* variable based on sensed congestion conditions.

9.2.6.1 CongestionSense state machine variables

classBAccessTime

classCAccessTime

See 9.2.2.

classBAccessDelay

Indicates the maximum amount of time any classB add traffic can wait to be transmitted before indicating congestion.

Range: [100 µs, 25.5 ms], in 100 µs increments

Default: 1.00 ms

classCAccessDelay

Indicates the maximum amount of time any classB add traffic can wait to be transmitted before indicating congestion.

Range: [100 µs, 25.5 ms], in 100 µs increments

Default: 1.00 ms

currentTime

See 9.2.2.

fwRate

localCongested

lpNrRate

maxAllowedRate

maxAllowedRate2

See 9.2.1.2.

myDualQueueStation

myTopoInfo.unreservedRate[*ri*]

See 9.2.2.

rateLowThreshold

Rate at or above which congestion on the outbound link is imminent.

Range: [0.5 * *rateHighThreshold*, 0.99 * *rateHighThreshold*]

Default: 0.9 * *rateHighThreshold*

stqDepth

See 9.2.2.

stqHighWatermark

The highest level of STQ occupancy since the last reset of this value.

stqLowThreshold

See 9.2.1.2.

stqLowWatermark

The lowest level of STQ occupancy since the last reset of this value.

9.2.6.2 CongestionSense state table

Table 9.4—CongestionSense state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	myTopoInfo.unreservedRate[myRI] < maxAllowedRate	1	maxAllowedRate2 = myTopoInfo.unreservedRate[myRI];	DEPTH
	—	2	maxAllowedRate2 = maxAllowedRate;	
DEPTH	stqDepth > stqHighWaterMark	3	stqHighWatermark = stqDepth;	LOW
	—	4	—	
LOW	stqDepth < stqLowWaterMark	5	stqLowWatermark= stqDepth;	QUEUE
	—	6	—	
QUEUE	myDualQueueStation == TRUE	7	—	SINGLE
	—	8	—	DUAL
SINGLE	lpNrRate > rateLowThreshold	9	localCongested = TRUE;	START
	(currentTime - classBAccessTime) >= classBAccessDelay	10		
	(currentTime - classCAccessTime) >= classCAccessDelay	11		
	—	12	localCongested = FALSE;	
DUAL	stqDepth > stqLowThreshold	13	localCongested = TRUE;	
	—	14	localCongested = FALSE;	

Row 9.4-3: Update the STQ occupancy high and low watermark values.
Row 9.4-7: Single-queue congestion conditions are tested.
Row 9.4-8: Dual-queue congestion conditions are tested.

(Optional: The SINGLE state and its conditions are necessary to support single-queue designs.)
Row 9.4-9: The consumption rate of unreserved traffic exceeds the *rateLowThreshold* limit.
Row 9.4-10: The classB access delay exceeds its specified limit.
Row 9.4-11: The classB access delay exceeds its specified limit.
Row 9.4-12: In the absence of congestion conditions, an uncongested condition is reported.

(Optional: The DUAL state and its conditions are necessary to support dual-queue designs.)
Row 9.4-13: The secondary transit queue has exceeded its low-threshold congestion-sensing limit.
Row 9.4-14: In the absence of congestion condition, an uncongested condition is reported.

9.2.7 FairnessThrottle state machine

The FairnessThrottle state machine updates appropriate variables based on sensed congestion conditions.

9.2.7.1 FairnessThrottle state machine variables

addRate
addRateCongested
addRateCongestedOk
addRateOk
admissionMethod
allowedRateCongested
See 9.2.1.2.
fwRate
See 9.2.1.2.
passAddFe
passAddFeOrStq
passAddFeCongested
See 9.2.2.
sendD
stqDepth
See 9.2.2.
stqHighThreshold
See 9.2.2.

9.2.7.2 FairnessThrottle state table

Table 9.5—FairnessThrottle state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	admissionMethod == RATE_BASED	1	—	RATES1
	—	2	—	SPAPE1
RATES1	addRate < allowedRate && **!**EntryInQueue(Q_TX_STQ)	3	addRateOk = TRUE;	RATES2
	addRate < allowedRate && fwRate > addRate && stqDepth < stqHighThreshold	4		
	—	5	addRateOk = FALSE; addRateCongestedOk = FALSE;	START
RATES2	addRateCongested < allowedRateCongested	6	addRateCongestedOk = TRUE	START
	—	7	addRateCongestedOk = FALSE	
SHAPE1	passAddFe && sendD && passAddFeOrStq;	8	addRateOk = TRUE;	SHAPE2
	—	9	addRateOk = FALSE; addRateCongestedOk = FALSE;	START
SHAPE2	passAddFeCongested	10	addRateCongestedOk = TRUE;	START
	—	11	addRateCongestedOk = FALSE;	

Row 9.5-1: The station uses the rate-based admissions method.
Row 9.5-2: The station uses the shaper-based admissions method.

(Optional: The RATES and RATE2 states are necessary to support the rate-based admission method.)
Row 9.5-3: Transmit if *addRate* is below its *allowedRate* limit, and no frame is found in the STQ.
Row 9.5-4: Transmit if *addRate* is below its *allowedRate* limit, the STQ traffic rates are sufficient, and the STQ depth is below the *stqHighThreshold*.
Row 9.5-5: Otherwise, fairness eligible transmissions are disallowed.

Row 9.5-6: Cannot transmit if *addRateCongested* has exceeded its *allowedRateCongested* limit.
Row 9.5-7: Transmit if congestion conditions do not apply.

(Optional: The SHAPE and SHAPE2 states are necessary to support the shaper-based admission method.)
Row 9.5-8: Transmit if shaper permissions are provided.
Row 9.5-9: Otherwise, fairness eligible transmissions are disallowed.

Row 9.5-10: Transmit if congestion-shaper permissions are provided.
Row 9.5-11: Otherwise, beyond-congestion transmissions are disallowed.

9.2.8 FairnessReceive state machine

9.2.8.1 FairnessReceive state machine literals

MAX_STATIONS
See 9.2.2.
FULL_RATE
SCALE
See 9.2.1.1.

9.2.8.2 FairnessReceive state machine variables

minWeight
The minimum weight of other attached stations.
myEdgeState
myMacAddress
myRI
myProtectMethod
See 9.2.2.
receivedHops
receivedRate
receivedScope
See 9.2.1.2.
ringInfo.totalHopsTx[*ri*]
See 9.2.2.
rxFrame
See 9.2.1.2.

9.2.8.3 FairnessReceive state machine routines

Hops(*frame*)
See 9.2.1.3.
Other(*ri*)
Min(*value1*, *value2*)
See 9.2.2.
ScaleDown(*value*)
ScaleUp(*value*)
See 9.2.1.3.

9.2.8.4 FairnessReceive state table

The FairnessReceive state table specification, as specified in Table 9.6.

Table 9.6—FairnessReceive state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	rxFrame.saCompact == myMacAddress && rxFrame.ri == myRI	1	receivedRate = ScaleDown(FULL_RATE, SCALE); receivedHops = MAX_STATIONS; receivedScope = 0;	RETN
	frame.saCompact==myMacAddress && myProtectMethod==CENTER_WRAP && (myEdgeState==INTO_EDGE || myEdgeState== FROM_EDGE)	2		
	rxFrame.ttl == 1	3		
	rxFrame.ri == Other(myRI)	4	receivedHops = ringInfo.totalHopsRx[Other(myRI)] + 1	FINAL
	—	5	receivedHops = Hops(rxFrame);	
FINAL	—	6	receivedRate = ScaleUp(rxFrame.fairRate, SCALE); receivedScope= Min(MAX_STATIONS, receivedHops + rxFrame.scope);	RETN

Row 9.6-1: If no other station is as congested, no congestion is reported.
Row 9.6-2: If no other wrapped station is as congested, no congestion is reported.
Row 9.6-3: After fairness frame timeouts, no congestion is reported.
Row 9.6-4: An other-side congestion point behaves as though the after-wrap station were congested.
Row 9.6-5: A same-side congestion point communicates the hops to congestion.
Row 9.6-6: The remaining congestion information is captured.

9.2.9 FairnessTransmit state machine

9.2.9.1 FairnessTransmit state machine definitions

MAX_STATIONS
See 9.2.2.
SCALE
See 9.2.1.1.

9.2.9.2 FairnessTransmit state machine variables

hops
A local copy of hops from the fairness-frame source.
localFairRate
lpFwRate
lpFwRateCongested
See 9.2.1.2.
minWeight
The minimum fairness weight of other stations.
myMacAddress
myRI
normFairRate
See 9.2.1.2.
receivedScope
See 9.2.1.2.
ringInfo.totalHopsTx[*ri*]
See 9.2.2.
rxFrame
See 9.2.1.2.
rxRate
A local copy of the received frame's *fairRate* value.
txFrame
The contents of a transmitted RPR frame.

9.2.9.3 FairnessTransmit state machine routines

Hops(*frame*)
See 9.2.1.3.
ScaleDown(*value*)
See 9.2.1.3.
TransmitFrame(*frame*)
See 9.2.2.

9.2.9.4 FairnessTransmit state table

The transmission of fairness frames involves updating congestion-point parameters, as specified in Table 9.7.

Table 9.7—FairnessTransmit state table

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

START	—	1	normFairRate = ScaleDown(localFairRate, SCALE);
			rxRate= rxFrame.fairRate;
			hops= Hops(rxFrame);
				FIRST

FIRST	rxRate >= normFairRate	2	txFrame.ttl = MAX_STATIONS;
			txFrame.ri = myRI;
			txFrame.saCompact = myMacAddress;
			txFrame.fairRate = normFairRate;
			txFrame.scope = receivedScope;
				FINAL

	hops <= ringInfo.totalHopsTx[myRI] && rxRate >= lpFwRateCongested/minWeight	3		
	rxRate >= lpFwRate/minWeight	4		
	receivedScope == 0	5		
	—	6	txFrame = rxFrame; txFrame.ttl = rxFrame.ttl – 1;	
FINAL	—	7	TransmitFairnessFrame(txFrame);	RETN

Row 9.7-1: Local values are precomputed for state machine clarity.
Row 9.7-2: If this station is more congested, this station's congestion level is reported.
Row 9.7-3: A closed-ring station becomes a tail if the *fairRate* received from the downstream neighbor is greater than or equal to the weight-adjusted rate of fairness eligible traffic transiting both the local station and the head of the congestion domain (*lpFwRateCongested*).
Row 9.7-4: If the ring is wrapped between the head station and the local station, the *lpFwRateCongested* does not maintain an accurate count of frames transiting both the local station and the head station. The *lpFwRate* is known to be larger than *lpFwRateCongested* as it includes all fairness eligible traffic transiting the local station. This value is taken as conservative alternative to the use of the *normLpFwRateCongested*.
Row 9.7-5: If the local station's fairness frame has returned, a zero-value *scope* tags the suspect information.
Row 9.7-6: Otherwise, the reported congestion information is passed upstream.

Row 9.7-7: The fairness frame is readied for transmission.

9.2.10 MultichokeTransmit state machine

At the expiration of a *reportingInterval*, the local station broadcasts a rate report to all stations on the ringlet.

9.2.10.1 MultichokeTransmit state machine definitions

FULL_RATE
See 9.2.1.1.
MAX_STATIONS
See 9.2.2.

9.2.10.2 MultichokeTransmit state machine variables

advertisementInterval
See 9.2.1.2.
currentTime
frame
See 9.2.2.
localCongested
See 9.2.1.2.
myMacAddress
myRI
See 9.2.2.
normFairRate
See 9.2.1.2.
reportCoef
A value indicating the number of *advertisingIntervals* that elapse between the sending of successive MCFFs.
Range: [8, 512]
Default: 10
reportedTime
The time at the start of the current multichoke-fairness report interval.
ringInfo.multichokeUser[*ri*]
See 9.2.2.

9.2.10.3 MultichokeTransmit state machine routines

MultichokeInd( )
See 9.2.1.3.
Other(*ri*)
ScaleDown(*value*)
See 9.2.1.3.
TransmitFrame(*frame*)
See 9.2.2.

9.2.10.4 MultichokeTransmit state table

Table 9.8—MultichokeTransmit state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	(currentTime - reportedTime) >= reportCoef * advertisementInterval	1	reportedTime = currentTime;	FIRST
	—	2	—	START
FIRST	**!**ringInfo.multichokeUser[myRI]	3	—	START
	localCongested	4	frame.fairRate = normFairRate;	FINAL
	—	5	frame.fairRate = ScaleDown(FULL_RATE, SCALE);	

FINAL	—	6	frame.ffType = MULTI_CHOKE;
			frame.saCompact = myMacAddress;
			frame.ttl = MAX_STATIONS;
			frame.ri = Other(myRI);
			TransmitFrame(frame);
			MultiChokeInd( );
				START

Row 9.8-1: Row 9.8-2: Operations are performed at the end of each *reportingInterval*.

Row 9.8-3: Rate reporting is suppressed if no station client uses the multi-choke fairness information.
Row 9.8-4: A congested station reports its locally computed *fairRate* to all stations on the ringlet.
Row 9.8-5: An uncongested station reports the FULL_RATE to all stations on the ringlet.

Row 9.8-6: Other fairness frame information is supplied.