9. Fairness

Editors' Notes: To be removed prior to final publication.

This document is a draft proposal for changes intended to clarify the fairness clause.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	Revised according to comments on D0.1 for TF review.
Draft 0.3, June 2002	Revised according to comments on D0.2 for WG review.
Draft 1.0, August 2002	Revised according to comments on D0.3 for TF review.
Draft 1.1, October 2002	Revised according to comments on D1.0 for WG review.
Draft 2.0 December 2002	Revised according to comments on D1.1 for TF review.
Draft 2.1, February 2003	Draft 2.1 for WG review, modified according to comments on D2.0.
Draft 2.2, April 2003	Draft 2.2 for WG ballot, modified according to comments on D2.1.

Per comment #596 March '03: The fairness adhoc will provide a recommendation on the use of FRTT by May '03 meeting. May03 #542 (540, 543, 505): adopt recommendation of FRTT adhoc.

Following to be added to clause 4 acronym list
LSFE: locally-sourced fairness eligible

Suggest global change of normLocalFairRate to fairRate

agingInterval (aggressive) state table to aggressive method rate computation
agingInterval (conservative) state table to conservative method rate computation

May03 #506: This document contains proposed improvements for fairness clarification.
May03 #517: All references to CSFE have been changed to LSFE.
May03 #636 (637, 638, 640): Populate PICS.

9.1 Overview

This clause specifies an algorithm for the computation of fair rates of ringlet access for locally-sourced fairness eligible (LSFE) traffic. The use of fair rates prevents one station from occupying a disproportionate

share of available ringlet capacity with respect to other stations on the ringlet. The fairness procedures reside within the MAC control sublayer of the MAC, as illustrated by the shaded region of Figure 9.1.

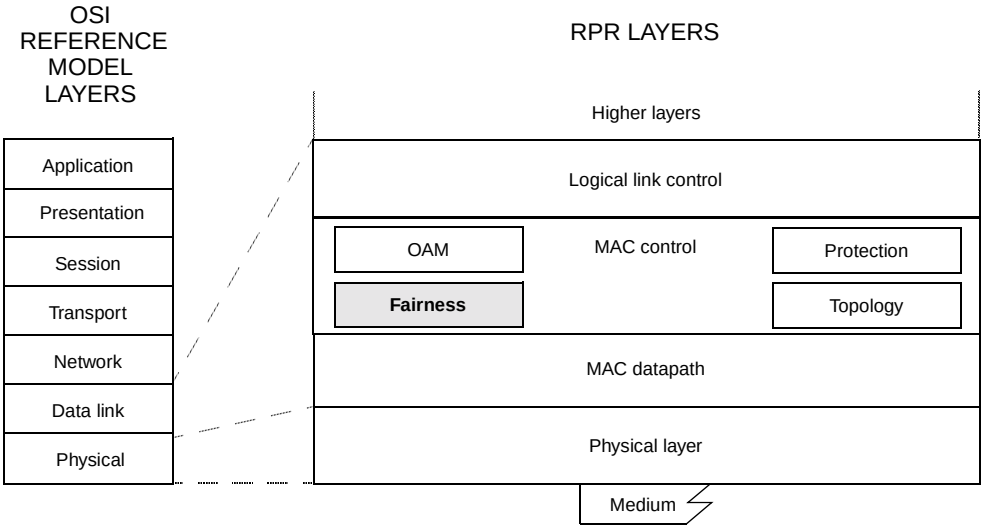

Figure 9.1—RPR layer diagram

A station normally contains two instances of the fairness algorithm. Each instance regulates traffic associated with one of the two ringlets. This is the case when a station is not in a wrapped state (i.e., is in a normal state with respect to wrapping) or is in the wrapped state and deploys an edge-wrapped implementation (6.9.3.8). A fairness instance is uniquely identified by the combination of the station address (*myMACAddress*) and the ringlet (*myRingletID)* carrying the data traffic whose rates are regulated by the fairness instance.

The exception is a station that is in the wrapped state and deploys a center-wrapped implementation (6.4.4.1). In this case, the fairness instance is uniquely identified by *myMACAddress*.

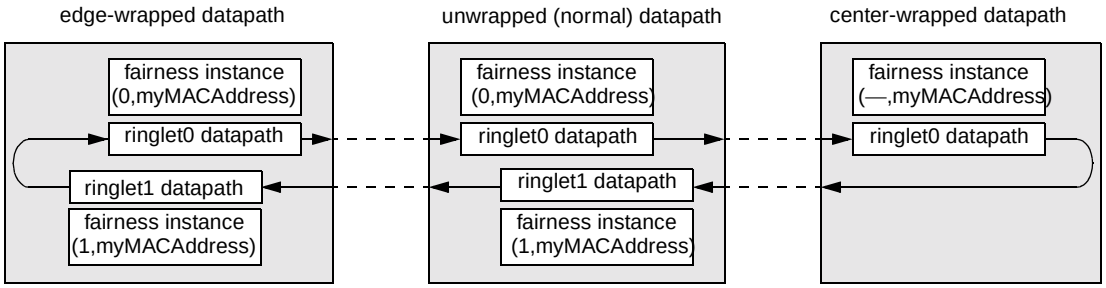

Figure 9.2—Fairness instances

NOTES
1—In order to simplify descriptions within this clause, ‘station’ is substituted for ‘fairness instance’ when it is clear from the context that the reference is to an individual fairness instance within the station. This is generally the case, since fairness procedures are performed independently on each ringlet.
2—This clause contains forward references to terms defined in 9.2 and identifiers defined by the fairness state machines of 9.3. The identifiers are italicized within the text.

9.1.1 Services and features

The fairness procedures described in this clause:

a) Support independent fairness operation per ringlet.
b) Carry control information on the ringlet opposing that of the associated data flow.
c) Regulate only classC and EIR classB (i.e., fairness eligible) traffic.
d) Compute fair rates associated with a source station (vs., for example, a source-destination station pair).
e) Separately regulate aggregate fairness eligible traffic added to the ringlet and that portion of added fairness eligible traffic that transits a point of congestion.
f) Scale fair rates in proportion to an administrative weight assigned to each fairness instance.
g) Allow ringlet capacity not explicitly allocated to be treated as available capacity.
h) Allow ringlet capacity explicitly allocated to subclassA1 or CIR classB, but not in use, to be treated as available capacity (i.e., bandwidth reclamation).
i) Support either single transit queue or dual transit queue deployment.
j) Support either the aggressive or the conservative rate adjustment method (i.e., one method must be supported).
k) Optionally report rate information to the MAC client.
l) Adjust fair rates within a few fairness round trip times (FRTTs) of the occurrence of a change in traffic load (i.e., fast response time).

Editors' Notes: *To be removed prior to final publication.*
May03 #518, 519 Proposed that the following two items be removed.

m) Converge to a steady-state fair rate within a reasonable period of time when presented with a steady input traffic pattern.
n) Behave predictably for all ringlet rates and ring diameters supported by the MAC.

9.1.2 Scope

This clause includes:

a) An overview (9.1).
b) Terms specific to the fairness algorithm (9.2).
c) The fairness state machine (9.3).
d) The fairness MIB attributes (9.4).
e) Fairness frame formats (9.5).
f) An explanation of ranges and default values (9.6)
g) An explanation of the effects of lost fairness frames (9.7).
h) An explanation of aging and rates (9.8).
i) PICS proforma (9.9).

Annex H of this document contains a C-code implementation of the fairness procedures.

9.1.3 Fairness algorithm overview

The fairness algorithm regulates the addition of fairness eligible (6.3) traffic to a ringlet in such a way that:

a) Congestion is controlled.
b) Throughput is minimally affected by congestion control activities.
c) Rate restrictions controlling congestion are applied fairly across stations contributing to congestion.

A station is congested when one or more of the following conditions is identified:

a) Occupancy of the secondary transit queue (6.4.2) is excessive.
b) The rate of transmission is excessive relative to the capacity of the transmission link.
c) Traffic is delayed excessively on the stage queue (6.8.5) while awaiting transmission.

Congestion is undesirable as it can result in a failure to meet the end-to-end commitments associated with service classes (6.3).

NOTE—The check of STQ occupancy is applicable only in the case of a dual-queue MAC. The check of transmission rate is always made in the case of a single-queue MAC but in the case of a dual-queue MAC is made only when the *checkRateThreshold* option is set. The check of delay on the stage queue is always made in the case of a single-queue MAC but in the case of a dual-queue MAC is made only when the *checkAccessDelay* option is set.

9.1.3.1 Regulating rates of traffic addition

Figure 9.5 shows a congested station (S6) and the set of contiguous stations (S1 - S6) contributing to the congestion at S6 (i.e., contributing stations).

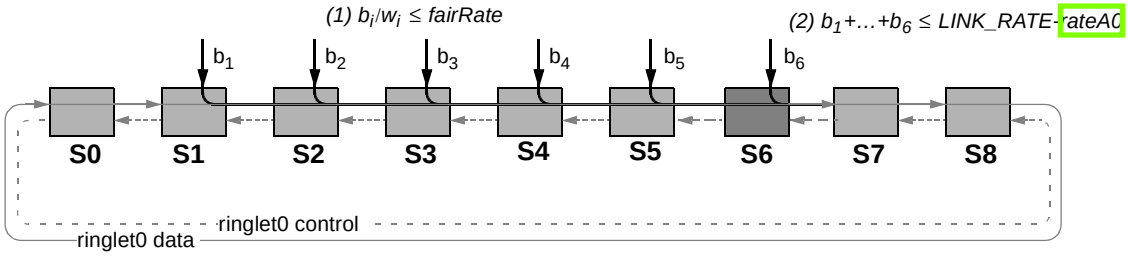

Figure 9.3—Congestion control objective

Each contributing station is associated with a rate (b_i) at which it adds fairness eligible traffic that also transits the congested station S6. The rate b_i is scaled by an administrative weight (w_i) that allows a station to add at a higher or lower rate than other stations without violating fairness principles. The objective of the fairness algorithm is to compute a *fairRate* applied to the downstream congested stations such that:

a) $\boldsymbol{b_i/w_i \le fairRate}$**:** Contributing stations limit their weight-adjusted rate so as not to exceed the *fairRate*.
b) $\boldsymbol{b_1+\ldots+b_6 \le LINK\_RATE - rateA0}$**:** The sum of fairness eligible traffic transmitted by the congested station maximizes use of available capacity without exceeding that capacity.

In order to meet the condition $b_i/ w_i \le fairRate$ at each contributing station, the *fairRate* computed by the congested station must be communicated to the contributing stations. Each station periodically sends an advertisement to its upstream neighbor. The advertisement can be propagated further upstream by placing information from the received advertisement in the next advertisement to be sent upstream. The congested station originates an advertisement that is propagated to the upstream contributing stations in this manner.

The advertisement allows each contributing station to limit its rate b_i to the current weight-adjusted *fairRate*, resulting in changes to rate statistics measured on the outbound link of the congested station. The rate statistics are used to adjust the *fairRate* in order to achieve the condition $b_1+\ldots+b_6 \le LINK\_RATE - rateA0$. The adjusted *fairRate* is advertised upstream. As illustrated in Figure 9.4, the feedback between the congested and contributing stations allows continuous adjustment of rates to meet both fairness and utilization conditions.

Figure 9.5 provides an example in which multiple stations on the ringlet are congested.

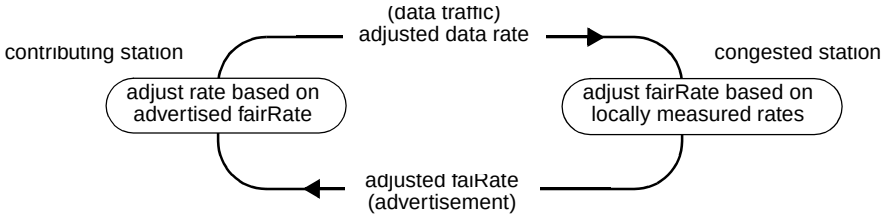

Figure 9.4—Congestion control feedback

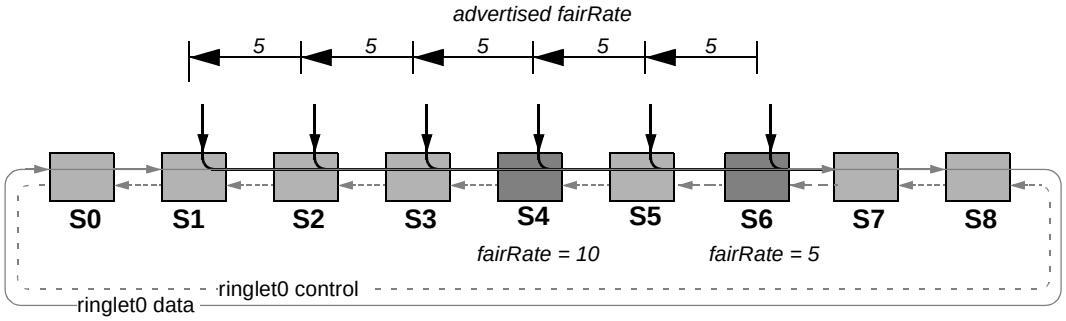

Figure 9.5—Received fairRate more restrictive than local fairRate

Each of the congested stations independently computes and advertises a *fairRate*. Congested station S4 computes a *fairRate* of 10 units and receives an advertised *fairRate* of 5 units originating from S6. Comparing the *fairRates*, S4 determines that the *fairRate* of S6 is more restrictive (i.e., smaller) than its own *fairRate*. S4 propagates the advertised *fairRate* of S6 rather than advertising its own *fairRate*. By advertising the more restricted *fairRate* of S6, it is ensured that the condition $b_i/w_i \leq fairRate$ is met for both sets of contributing stations (i.e., S1 to S3, and S4 to S5). This example also illustrates that a congested station does not necessarily advertise its locally computed *fairRate*. It does not do this when it receives an advertisement carrying a more restrictive *fairRate* from a downstream station.

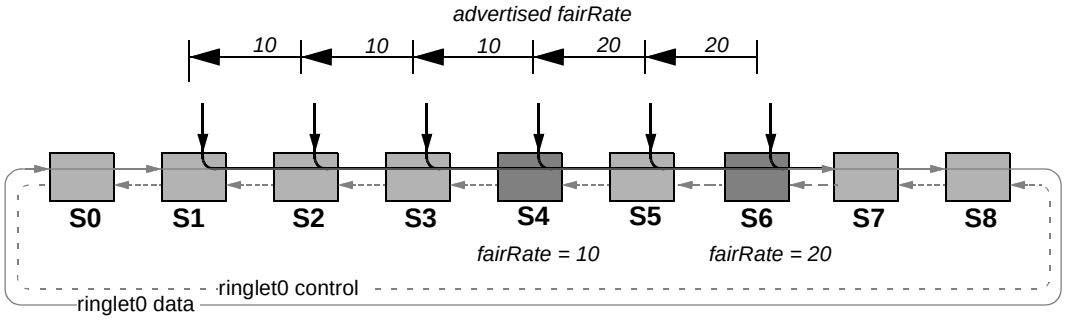

Figure 9.6—Local fairRate more restrictive than received fairRate

Figure 9.6 illustrates the case in which the congested stations are again S4 and S6 but the *fairRate* computed by S4 is more restrictive (i.e., smaller) than the *fairRate* computed by S6. Congested station S4 computes a *fairRate* of 10 units and receives an advertised *fairRate* of 20 units originating from S6. Comparing the *fairRates*, S4 determines that its *fairRate* is not larger than that of S6. S4 advertises its own *fairRate* rather than propagating the advertised *fairRate* of S6. By advertising the more restrictive *fairRate* of S4 between S4 and S1, and the less restrictive *fairRate* of S6 between stations S6 and S4, it is ensured that the condition $b_i/w_i \leq fairRate$ is met for both sets of contributing stations (i.e., S1 - S3 and S4 - S5), while not unnecessarily restricting the rates of stations S4 and S5.

As illustrated in Figure 9.7 a station can also inform its upstream neighbor that upstream stations are not contributing to downstream congestion. In this case, the station advertises a value to the upstream neighbor indicating that the neighbor can send at the full rate available to fairness eligible traffic. Station S1 is the final station in a sequence of stations contributing to congestion at S4. S1 advertises to S0 that traffic received from S0 is not contributing to downstream congestion and that S0 can send at the full rate. Station S7 illustrates that a station receiving a full rate advertisement will propagate that advertisement further upstream if it is not congested.

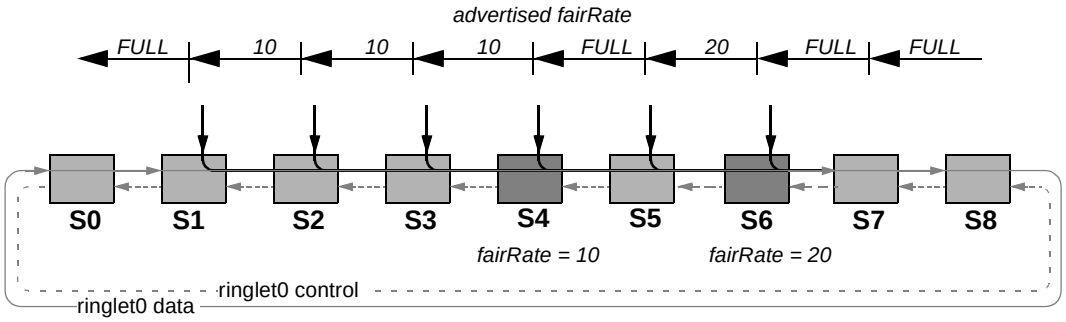

Figure 9.7—Advertising the full rate

As illustrated in the preceding examples, a station can advertise one of three possible values to its upstream neighbor:

a) Its *fairRate*.
b) The *fairRate* advertised by its downstream neighbor.
c) A value indicating that upstream stations are not contributing to downstream congestion.

An advertisement identifies its station of origin. If the advertisement carries the local *fairRate* or the full rate value, the origin is the local sending station. If the advertisement propagates the *fairRate* received from the upstream neighbor, the origin is the station whose *fairRate* is carried by the advertisement. The advertisement also carries a time-to-live field that is assigned the value MAX_STATIONS by the originating station.

As illustrated by Figure 9.5, a station advertising its locally computed *fairRate* is said to lie at the head of a congestion domain. The downstream link of the head station is known as the congestion point. The congestion domain extends upstream from the head through all stations that propagate the advertisement until reaching a station that does not. The station at the upstream end of the congestion domain is known as the tail. Station S1 is the tail of the congestion domain A. It receives from S2 the advertisement originated by the congestion point S3. S1 terminates congestion domain A by advertising the FULL_RATE further upstream to S0. Station S3 illustrates a case in which the tail of downstream congestion domain B is also the head of upstream congestion domain A. Station S3 receives an advertisement originated by S4 but advertises its locally computed *fairRate* upstream to S2.

It is a property of the congestion domain that the *fairRate* of the head is less than (i.e., more restrictive than) any other station within the congestion domain. All other stations in the congestion domain are contributing stations with respect to the congestion point. The contributing stations may, or may not, be congested. The *fairRate* of the congestion point is propagated hop-by-hop from the head station to the tail station. Each contributing station in the congestion domain sets its rate b_i based on the *fairRate* of the head. The FULL_RATE is advertised between the tail (i.e., end) of one domain and the head (i.e., beginning) of the next upstream domain or on all links in the absence of congestion domains. Stations that are not contributing stations within a congestion domain (i.e., are the head of one domain but not the tail of another or do not lie within a congestion domain) are not limited by the rate b_i.

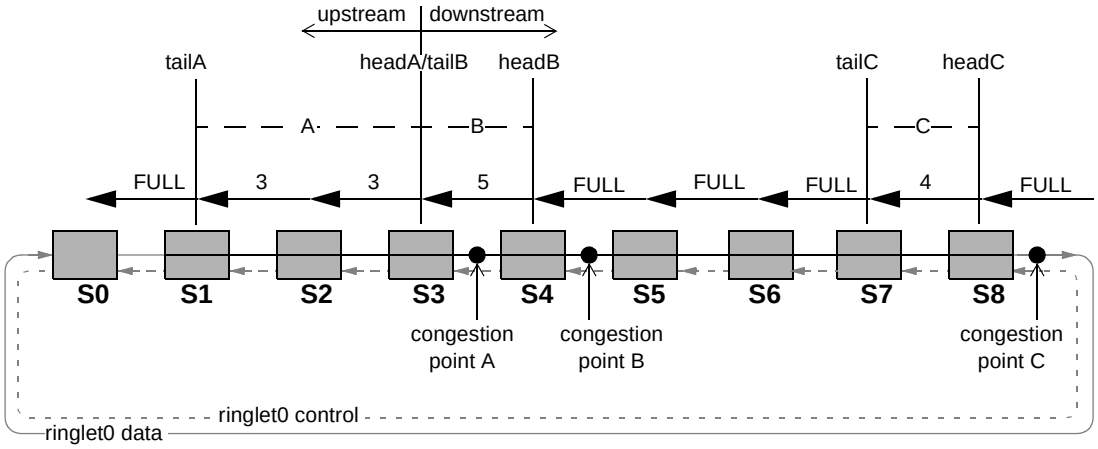

Figure 9.8—Congestion domains

For contributing stations, rate b_i is assigned the weight-adjusted value of the most recently received *fairRate* (i.e., the weight-adjusted *fairRate* of the congestion point). For non-contributing stations, rate b_i is allowed to increase asymptotically (ramp) towards the maximum allowed value of b_i. The ramping of b_i prevents the oscillation in traffic rates that would occur if traffic increased suddenly at a station that had recently been a congestion point.

The value b_i is accompanied by the value of *hopsToCongestion*, which is the distance, in hops, between a station S_i and its associated congestion point. This value is required in order to determine whether a data frame is subject to the rate restriction of b_i. On arrival of an advertisement at station S_i, the *hopsToCongestion* is computed as the difference between the TTL value inserted at the origin station and the TTL value of the received advertisement.

NOTE—An exception to this method of computation is made when the most recently received advertisement originated on the opposing ringlet and the ring is in the wrapped state (*ringTopologyType == RING_OPEN*). In this case, the *hopsToCongestion* is set to the number of hops between the local station and the downstream wrap point (*hopsToDsWrap*). This value is maintained by the protection protocol (see Clause 10).

In addition to restricting locally added fairness eligible traffic transiting the congestion point, the station S_i also restricts the aggregate fairness eligible traffic (a_i) added by station S_i (i.e., without regard to whether the traffic transits a congestion point). Rate a_i is restricted to a configured maximum rate for S_i, not exceeding the unreserved rate of the ringlet. The rate a_i can be assigned a value smaller than the maximum rate.

The rates a_i and b_i are known within the fairness algorithm as the *allowedRate* and the *allowedRateCongested*, respectively. A station periodically recomputes its *fairRate*, *allowedRate*, *allowedRateCongested*, and *hopsToCongestion*. The period between computations is known as the *agingInterval* since one of the computations performed is the aging of rate counters. The length of the *agingInterval* is based on the data rate of the ring and has the same value at all stations on the ring.

Following their computation, the *allowedRate*, *allowedRateCongested*, and *hopsToCongestion* are optionally transferred to the MAC client via an MA_CONTROL.indication having an opcode of SINGLE_CHOKE_IND (see Table 5.3). The information can be used by the client to perform fairness activities beyond the scope of the MAC (an example of which is provided in Annex J.3).

The *fairRate* computed by the station is reported by broadcast to all other stations on the ringlet using a multi-choke fairness frame (9.5.1). The time between successive broadcasts of a station's *fairRate* is known as the *reportingInterval*, and is a configured multiple of the *advertisingInterval*. On learning the *fairRate* of a station, or on broadcasting its own *fairRate*, the *fairRate* is optionally transferred to the MAC client via an

MA_CONTROL.indication having an opcode of MULTI_CHOKE_IND (see Table 5.3). The information can be used by the client to perform fairness activities beyond the scope of the MAC.

9.1.3.2 Rate normalization

A rate communicated from one station to another is normalized in order to (1) ensure that the rate is uniformly interpreted by stations on the ringlet and (2) scale the rate value to allow it to be efficiently encoded as an integer value within the 16-bit *fairRate* field (see 9.5.3) of the fairness frame. Normalization of a rate is performed by dividing the locally significant rate by a normalization coefficient (*normCoef*). The *normCoef* is the product of the following values:

localWeight: Normalizes the rate consistent with a standard localWeight value of one.
rateCoef: Normalizes the rate consistent with a standard LINK_RATE of 2.5 Gbps.
ageCoef: Normalizes the rate consistent with a standard unit of bytes-per-agingInterval.

That is, normCoef = localWeight * rateCoef * ageCoef;

When received by a station, a normalized rate may be:

a) Converted to a locally significant rate (i.e., localized) through multiplication by the local *normCoef*.
b) Maintained in the normalized form for propagation to other stations.
c) Compared to a local rate that has been normalized.

A normalized rate can be converted to a locally significant rate by multiplying the rate by the *normCoef* of the local station.

9.1.3.3 Measurement of traffic rates

As previously described by Figure 9.4, the *fairRate* computed locally by a station is adjusted based on traffic measurements made by the station..

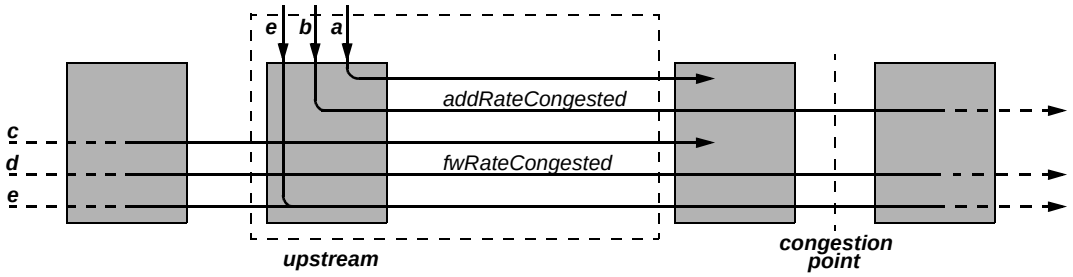

Figure 9.9—Measured rates

In Figure 9.9, the arrows lettered (a) through (e) represent traffic as follows:

a) Fairness eligible traffic added by the local station bound for a destination station that is not beyond the congestion point.
b) Fairness eligible traffic added by the local station bound for a destination that is beyond the congestion point. The rate of this traffic is the *addRateCongested*.
c) Fairness eligible traffic transiting the local station and bound for a destination that is not beyond the congestion point.
d) Fairness eligible traffic transiting the local station and bound for a destination that is beyond the congestion point. The measured rate of this traffic is the *fwRateCongested*.

e) Traffic of classB within CIR or of subclassA1 transmitted (i.e., added or transited) by the local station

In Figure 9.10, the circled groups of arrows represent traffic as follows:.

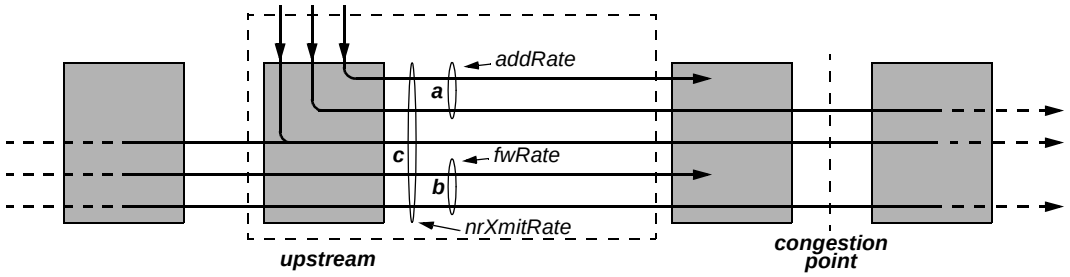

Figure 9.10—Measured rates

a) The total fairness eligible traffic added by the local station. This rate is maintained by the rate counter *addRate*.
b) The total fairness eligible traffic transited by the local station. This rate is maintained by the rate counter *fwRate*.
c) The total traffic transmitted by the local station excluding traffic of subclassA0. This rate is maintained by the rate counter *nrXmitRate*.

NOTE—As specified in 8.2.2.2, a frame with frame.sc == CLASS_A0 or frame.sc == CLASS_A1 is not fairness eligible and *frame.fe* field is ignore

Each byte added or transited by the station is evaluated to determine which, if any, of the five rate counters (*addRate*, *addRateCongested*, *fwRate*, *fwRateCongested*, and *nrXmitRate*) are to be incremented. The update of rate counters can be delayed until up to 256 bytes are available for counting, allowing more efficient maintenance of rates.

Prior to the use of a measured rate in the *fairRate* calculation, the rate is smoothed with respect to past rate measurements. This is done by computing a weighted average of the current rate value and the last smoothed rate value. The weight coefficient associated with the weighted average is a configured parameter. This smoothing method is called low pass filtering. The current smoothed rates are maintained separately from the rate counters. The smoothed rates are named *lpAddRate*, *lpAddRateCongested*, *lpFwRate*, *lpFwRateCongested*, and *lpNrXmitRate*, respectively. The rate counters are themselves unchanged by low pass filtering.

After the rate counters have been referenced in the computation of smoothed rates, an aging operation is applied to the rate counters. During the period between aging operations, known as an *agingInterval*, each rate counter maintains a byte count. Aging converts the byte counts to values that asymptotically approach the data rate, assuming that the rate of offered traffic on the ringlet is stable over a period of time.

In addition to maintaining the rate counter values as rates, aging has the effect of preventing the overflow of the rate counter and smoothing the value of the rate counter with respect to previous rate counter values. A rate counter is aged by multiplying its value by the expression (*ageCoef*-1)/*ageCoef*. The *ageCoef* specifies the relative weights assigned to (a) the change in value of the rate counter during the most recent *agingInterval* and (b) the value of the aged rate counter at the expiration of the previous *agingInterval*. The use of aging to derive rates is further explained in 9.8.

9.1.3.4 Computation of policing indications

The allowed rates of fairness eligible traffic (9.1.3.1) and the measured rates (9.1.3.3) are used to maintain boolean policing indications as follows:

a) The *addRateOK* indication which specifies whether a byte of a fairness eligible frame (i.e., FE-marked frame) is allowed to be transferred from the MAC client to the MAC.
b) The *addRateCongestedOK* indication which specifies whether a byte of a fairness eligible frame bound for a destination beyond the congestion point is allowed to be transferred from the MAC client to the MAC.

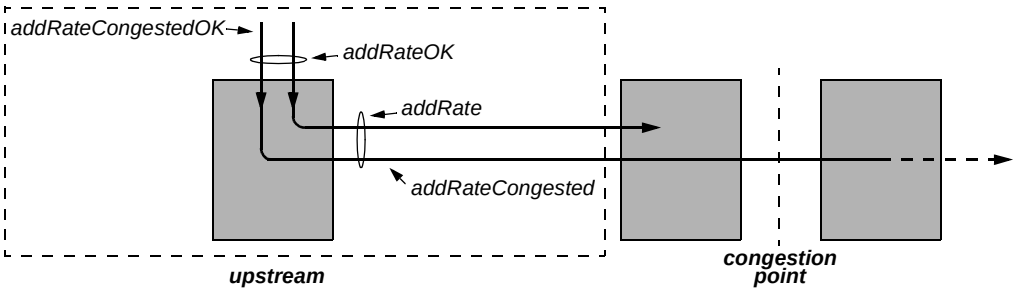

Figure 9.11—Policing indications

As illustrated by Figure 9.11, *addRateOK* and *addRateCongestedOK* regulate the transfer of traffic that would be included in the *addRate* and the *addRateCongested*, respectively. The indication *addRateOK* is set to allow traffic to pass if the measured *addRate* does not exceed the current *allowedRate* and the station is determined to have sufficient downstream capacity to serve upstream stations. Sufficient downstream capacity is available when either of the following conditions is met:

a) Upstream stations are not in need of more downstream capacity since traffic is not accumulating in the secondary transit queue (STQ).
b) Upstream stations are not 'starved' for downstream capacity and the station is not fully congested with respect to STQ occupancy.

The indication *addRateCongestedOK* is set to allow traffic to pass if the addRateOK conditions have been met and the *addRateCongested* does not exceed the *allowedRate*.

The policing indications are evaluated after a byte, or a group of up to 256 bytes, has been counted. The policing indications are referenced by the datapath in the evaluation of the sendC indication (see 6.5.4).

9.1.3.5 Adjusting the fairRate

As previously described in 9.1.3.1, a station adjusts its *fairRate* at each *agingInterval*. The station must use one of the following two methods of rate adjustment:

a) **Aggressive**: Provides responsive adjustments that favor utilization of capacity over rate stability.
b) **Conservative**: Provides highly damped adjustments that favor rate stability over utilization of capacity.

Stations using different methods can interoperate on the ring.

The aggressive method is the simpler of the two and is described as follows:

a) **The station ~~has just become~~ congested**: The *fairRate* is set to the *lpAddRate*. As previously described in 9.1.3.3, the *lpAddRate* is the smoothed measured rate of addition to the ringlet. If the station is the head of a congestion domain, each upstream station within the congestion domain will ~~adjust its *fairRate* to the lpAddRate~~ of the head station.

b) ~~**The station continues to be congested**: The *fairRate* is unchanged. This condition occurs when the *fairRate* has already been reduced to the *lpAddRate*, but sufficient time has not passed to allow advertisement of the rate to remedy the congestion.~~

c) **The station is not congested**: The *fairRate* is set to the *unreservedRate*. The *unreservedRate* is the capacity that remains after reserving capacity for traffic of subclassA0 and is the maximum value to which the *fairRate* can be set.

The conservative method differs from the aggressive method in that a station can remain in a congested state after the station is no longer congested. This provides hysteresis in the transition from congested to UNCG state and prevents the rate oscillation that might occur if large amounts of traffic transited a station that was recently congested. In some cases, the conservative method requires that the *fairRate* not be adjusted until sufficient time has passed to ensure that the effect of any previous adjustment has been observed. This period is known as the fairness round trip time (FRTT). The conservative method is described as follows:

a) **The station has just become congested and enters a congested state**: The *fairRate* is assigned a weighted fair share of the *unreservedRate*. The capability of computing a weighted fair share is optional. Stations without this capability set the *fairRate* to the *lpAddRate*, as in the aggressive method.

b) **The station was in the congested state, the station is currently congested, and an FRTT has elapsed since the last *fairRate* adjustment**: The *fairRate* is ramped down, or reduced by a fraction of its current value. ramping donwnl reduces congestion gradually in contrast to aggressive rate adjustment which reduces the *fairRate* directly to the *lpAddRate*.

c) **The station was in the congested state, the station is not currently congested, and an FRTT has elapsed since the last *fairRate* adjustment**: The *fairRate* is ramped up, or increased by a fraction of the difference between its current value and the *unreservedRate*. ramping up increases rates gradually in contrast to aggressive rate adjustment which increases the *fairRate* directly to the *unreservedRate*.

d) **The station is in a congested state and the *fairRate* is close to or above the *unreservedRate* (i.e., is fully ramped up)**: The *fairRate* is set to the *unreservedRate* and the station enters an UNCG state. This is in contrast to the aggressive method in which the *fairRate* is set to the *unreservedRate* as soon as the station is no longer congested.

e) **The station is seriously congested due to high STQ occupancy**: The *fairRate* is assigned a weighted fair share of the sum of the *lpAddRate* and *lpFwRate*, provided that value is less than the current *fairRate*. The capability of computing a weighted fair share is optional. Stations without this capability set the *fairRate* to the *lpAddRate*, provided that value is less than the current *fairRate*. This condition does not occur in the aggressive method as the *fairRate* is immediately reduced when congestion occurs.

Stations deploying the aggressive and conservative methods interoperate on the ringlet. A station must implement one of the two methods. Both methods converge to the *fairRate* value described in 9.1.3.1 when offered traffic at stations on the ringlet is constant.

9.1.3.6 Computing the allowedRate

As illustrated by Figure 9.11, the *allowedRate* specifies the aggregate rate at which locally-sourced fairness eligible traffic can be added to the ringlet by the local station. As in the case of the *fairRate* computation (9.1.3.5), the method of computing the *allowedRate* depends on the method of rate adjustment deployed by the station. In the case of aggressive rate adjustment, the *allowedRate* is always assigned the configured maximum rate (*maxAllowedRate*) of the station.

In the case of conservative rate adjustment, the *allowedRate* is computed as follows:

a) **The station is in the uncongested state:** The *allowedRate* is ramped up towards the *maxAllowedRate*. This is in contrast to the aggressive method in which the *allowedRate* is directly set to the *maxAllowedRate*.
b) **The station is in the congested state:** The *allowedRate* is assigned the locally computed *fairRate* (or the *maxAllowedRate* if the *fairRate* exceeds the *maxAllowedRate*).

When using the conservative method, the hysteresis associated with the transition from the congested to the uncongested state prevents the rate oscillation that might occur if the *allowedRate* was permitted to increase suddenly and cause congestion at one or more stations. This oscillation is not an issue when using the aggressive method since the *allowedRate* is constant.

9.1.3.7 Computing the allowedRateCongested

The *allowedRateCongested* specifies the rate at which locally-sourced fairness eligible traffic destined for stations beyond the congestion point can be added to the ringlet. Unlike the *fairRate* and *allowedRate* computations, the computation of the *allowedRateCongested* does not depend on the rate adjustment method. In the presence of downstream congestion, the *allowedRateCongested* is set to the value of the *fairRate* carried by the most recently received rate advertisement. The received *fairRate* is the locally computed *fairRate* of the downstream congestion point. This assignment reflects the rule that a station lying within a congestion domain (i.e., contributing station) is not permitted to add fairness eligible traffic transiting the congestion point at a rate greater than the *fairRate* of the congestion point. In the absence of downstream congestion, the *allowedRateCongested* is ramped up, allowing stations to effectively use available capacity until congestion is encountered.

9.1.3.8 The fairness round trip time (FRTT)

Editors' Notes: *To be removed prior to final publication.*
To be supplied by FAH by May '03 consistent with Mar '03 #596.

9.2 Terminology specific to the fairness algorithm

This section describes terminology specific to the fairness algorithm. See 9.3 for definitions of identifiers used within fairness state machines.

9.2.1 aggressive rate adjustment: A method of rate adjustment in which the rate can be further adjusted before the effect of the previous rate adjustment is observed. See 9.1.15 for further description.

9.2.2 aging: A method of periodically adjusting a counter such that the value asymptotically approaches a rate and the maximum value of the counter is bounded. See section 9.1.18.4 for further description.

9.2.3 congestion domain: The set of contiguous links traversed by a single choke fairness frame indicating congestion (i.e., rate field value less than FULL_RATE) such that the value of the source address is not modified.

9.2.4 congestion point: A station identified by a local station as the head of a congestion domain containing the local station.

9.2.5 conservative rate adjustment: A method of rate adjustment in which the rate is not further adjusted until sufficient time has passed for the effect of the previous rate adjustment to be observed, except in cases of excessive STQ occupancy.

9.2.6 fairness frame: A type of frame used by the fairness algorithm to advertise or report *fairRates*. See 9.5 for a description of the fairness frame format.

9.2.7 head: A station that advertises its locally computed non-FULL_RATE *fairRate* to its upstream neighbor. A station that is the head of an upstream congestion domain can also be the tail of a downstream congestion domain.

9.2.8 localize: To adjust a rate value to allow correct local interpretation of that rate. See 9.1.18 for further details.

9.2.9 multi choke: Related to the visibility of multiple points of congestion on the ringlet. The MAC does not deploy a multi choke fairness algorithm but it does distribute multichoke rate information on the ringlet and optionally reports such information to the MAC client.

9.2.10 multi choke fairness frame : A fairness frame reporting the *fairRate* of a station to all stations on the ringlet.

9.2.11 normalize: To adjust a local value to allow a global interpretation when the value is communicated to other stations on the ringlet. See 9.1.18 for further details.

9.2.12 ramping: A method of gradually changing a rate employing a coefficient that is configured to control the degree of change.

9.2.13 single choke: Related to the visibility of only a single point of congestion on the ringlet. The MAC deploys a single choke fairness algorithm as each station has visibility to no more than one point of congestion on the ringlet.

9.2.14 single choke fairness frame (SCFF): A fairness frame carrying a *fairRate* advertisement from a station to its upstream neighbor.

9.2.15 tail: A station that receives an advertisement containing a non-FULL_RATE *fairRate* and sends an advertisement containing a fairRate that is not equal to the fairRate it receives. A station that is the tail of a downstream congestion domain can also be the head of an upstream congestion domain.

9.2.16 weighted fairness: The use of a weight associated with each fairness instance allowing a station to add at a higher or lower rate than other stations without violating fairness objectives. Weighted fairness can be effectively disabled by assigning equal weights to fairness instances.

9.2.17 weighted fair share: The ratio of the weight of a station to the sum of the weights of stations active on the ringlet.

9.3 Fairness state machine

The fairness state machine description is divided into the following state tables:

a) per-byte activities.
b) per-*agingInterval* activities.
c) aggressive rate adjustment.
d) conservative rate adjustment.
e) per-*advertisingInterval* activities.
f) per-*activeWeightsComputationInterval.*
g) per-*reportingInterval* activities.
h) fairness frame receive processing.

The fairness state machine shall implement the state tables specified in this section and meet the corresponding state table interface requirements, also specified here. In the case of any ambiguity between

the text and the state tables, the state tables shall take precedence. The notation used in the state machine is described in 3.4.

NOTE—Implementations may choose any conforming means of realizing the state tables described herein, provided that the external behavior of the fairness state machine is unchanged.

9.3.1 Inputs

classBAccessDelayTimerExpired
: Provided by Clause 6 to indicate whether or not the classB access delay timer has expired. Allowed values are {TRUE; FALSE}.

classCAccessDelayTimerExpired
: Provided by Clause 6 to indicate whether or not the classC access delay timer has expired. Allowed values are {TRUE; FALSE}.

frame
: The frame made available by the datapath for fairness processing.

hopsToDsWrap
: Provided by Clause 10 to indicate the number of hops from the local station to the current downstream wrap point.

myMACAddress
: The address of the local station MAC.

myRingletID
: The ringlet ID of the ringlet processing the received frame. Allowed values are {RINGLET_0, RINGLET1}

myWrappingMethod
: Wrapping method of the local station. Allowed values are {EDGE_WRAP, CENTER_WRAP}.

numStations
: Provided by Clause 10 to indicate number of stations associated with the ring.

rateA0
: See 6.2.1.

rateA1
: See 6.2.1.

rateB
: See 6.2.1.

ringTopologyType
: Wrapping state of the ring. Allowed values are {RING_CLOSED, RING_OPEN.

stqDepth
: Provided by Clause 6 to indicate number of bytes currently occupied within the STQ.

weights[]
: An array of integer type and size MAX_STATIONS provided by Clause 10 to indicate the *localWeight* associated with each station on the ringlet. This array is referenced during the computation of *activeWeights* at the expiration of an *activeWeightsInterval*. The array is referenced only in the case of conservative rate adjustment.

9.3.2 Constants

FULL_RATE
: A special value of *fairRate*, indicating the absence of congestion. The value of this constant is $FFFF_{16}$.

MAX_STATIONS
: See 3.2.

MAX_WEIGHT
: The maximum administrative weight that can be assigned. The value of this constant is 255.

9.3.3 System values

dualQueueMAC
: A boolean value indicating whether or not the station deploys a dual-queue MAC.

LINK_RATE
: The rate of the link in units of bytes per *ageCoef agingIntervals*. A value of LINK_RATE is associated with each PHY type and is specified in the annex describing the reconciliation sublayer associated with that PHY.

9.3.4 Global variables

The variables listed in this section are global variables referenced by the fairness procedures and defined in 3.2.

allowedRate
: See 3.2.

allowedRateCongested
: See 3.2.

hopsToCongestion
: See 3.2.

activeWeightsCoef
: A value indicating the number of *agingIntervals* that elapse between successive computations of the activeWeights. The allowed range is [8, 512]. The default value is 10.

addRateOK
: Indicates whether LSFE traffic is allowed to be transmitted to the ringlet from an add queue of the local station. A value of TRUE indicates that sending is allowed.

addRateCongestedOK
: Indicates whether LSFE traffic bound for a destination beyond the congestion point is allowed to be transmitted to the ringlet from an add queue of the local station. A value of TRUE indicates that sending is allowed.

9.3.5 Configured variables

The variables listed in this section contain values provided by the MAC Layer Management Entity (MLME) (Clause 12). The values provided shall be consistent with the specified ranges and defaults.

activeWeightsCoefficient
: A value indicating the number of *advertisingIntervals* that elapse between successive computations of activeWeights. The allowed range is [8, 512]. The default value is 10.

activeWeightsDetection
: A value indicating whether activeWeights is computed each activeWeightsInterval. This option is applicable only to conservative rate adjustment. Allowed values are {TRUE, FALSE}. The default value is FALSE.

advertisementRatio
: The ratio of (a) link capacity reserved for fairness frames to (b) LINK_RATE. Allowed values are in the range [0.01, 0.00025]. The default value is 0.00125.

ageCoef
: The coefficient used by the aging procedure to specify the relative weights assigned to (a) the change in the value of a rate counter during the most recent *agingInterval* and (b) the value of the rate counter at the end of the previous *agingInterval*. Allowed values are {1, 2, 4, 8, 16}. The default value is 4.

checkAccessDelayTimer
: A value indicating whether the values of *classBAccessDelayTimerExpired* and *classCAccessDelayTimerExpired* should be considered during local congestion evaluation in a

dual-queue implementation (*dualQueueMac*). Allowed values are {TRUE, FALSE}. The default value is FALSE.

checkRateThreshold

A value indicating whether total non-reserved traffic transmitted (*lpNrXmitRate*) should be compared to the *rateLowThreshold* during local congestion evaluation in a dual-queue implementation (*dualQueueMac*). Allowed values are {TRUE, FALSE}. The default value is FALSE.

localWeight

An administrative weight assigned to each fairness instance to permit the scaling of *fairRate* values among stations on the ringlet. This allows one station to use a larger share of available capacity than another station without violating fairness principles. The allowed range is [1, 255]. The default value is 1.

lpCoef

The coefficient used by the low pass filter procedure to specify the relative weights applied to (a) the increase in the rate-count value during the most recent *agingInterval* and (b) the previous low pass filtered rate. The former is assigned a weight of 1 and the latter a weight of (*lpCoef*-1). Allowed values are {16, 32, 64, 128, 256, 512}. The default value is 64.

maxAllowedRate

The maximum permitted value of the *allowedRate*. The range is [1, LINK_RATE]. The default value is the LINK_RATE.

rampCoef

The coefficient used for ramping a rate. Allowed values are {16, 32, 64, 128, 256, 512}. The default value is 64.

rateHighThreshold

Rate at or above which congestion on the outbound link is declared. The range is [0.4 * *unreservedRate*, 0.99 * *unreservedRate*]. The default value is 0.95 * *unreservedRate*.

rateLowThreshold

Rate at or above which congestion on the outbound link is imminent. The range is [0.5 * *rateHighThreshold*, 0.99 * *rateHighThreshold*]. The default value is 0.9 * *rateHighThreshold.*

reportCoef

A value indicating the number of *advertisingIntervals* that elapse between the sending of successive MCFFs. The allowed range is [8, 512]. The default value is 10.

stqFullThreshold

A level of STQ occupancy at or above which the STQ is almost full. The range is [*stqHighThreshold* + *sizeMTU*, *sizeSTQ* - *sizeMTU*]. The default value for *stqFullThreshold* is *sizeSTQ* - 2**sizeMTU*.

stqHighThreshold

A level of STQ occupancy at or above which LSFE frames are no longer admitted. Defined only for a dual transit-queue implementation. The allowed range is [3**sizeMTU*, *stqFullThreshold* - *sizeMTU*]. The default value is 0.25**stqFullThreshold*.

stqLowThreshold

A level of STQ occupancy at or above which congestion on the outbound link is imminent. Defined only for dual transit-queue implementations. The range is [*sizeMTU*, *stqMedThreshold* - *sizeMTU*]. The default value is 0.5**stqHighThreshold.*

stqMedThreshold

A level of buffer occupancy in a dual-queue deployment, at or above which congestion on the outbound link is declared. The allowed range is [*stqLowThreshold* + *sizeMTU*, *stqHighThreshold* - *sizeMTU*]. The default value is 0.5 * (*stqHighThreshold* + *stqLowThreshold*).

9.3.6 Variables (calculated)

activeWeightsInterval

The time between successive computations of activeWeights. Applicable only in the case of conservative rate adjustment and *activeWeightsDetection*.

activeWeights

A value nominally representing the sum of the weights of stations contributing fairness eligible traffic to the ringlet. The value is in the range [1, MAX_WEIGHT * MAX_STATIONS] inclusive.

addRate

The rate, in units of bytes per *ageCoef agingIntervals*, at which fairness eligible traffic is transmitted to the ringlet from the add queues of the local station.

addRateCongested

The rate, in units of bytes per *ageCoef agingIntervals*, at which fairness eligible traffic intended for destinations downstream of the congestion point, is transmitted to the ringlet from the add queues of the local station.

advertisedFairRate

The value placed in the *fairRate* field of an SCFF sent by the local station. The value is either the *normLocalFairRate* a local station that is a congestion point, the *normLocalFairRate* of an downstream stream congestion point as learned from a received SCFF, or the value FULL_RATE indicating that the local station does not lie within a congestion domain.

advertisingInterval

The interval at which SCFFs are sent. The *advertisingInterval* is computed as *sizeFF* / (LINK_RATE * *advertisementRatio*).

agingInterval

The interval at which the *localFairRate* computation is performed. The *agingInterval* is fixed for a particular value of the LINK_RATE. The *agingInterval* is 100μsec for ring rates of 622Mbps. and above and 400μsec for a ring rates below 622Mbps, as illustrated by Table 9.2.

Table 9.1—The agingInterval as a function of ring rate

ring rate	agingInterval
≥ 622Mbps.	100 usec
< 622Mbps.	400 usec

downstreamCongested

A boolean value indicating whether or not the local station has detected downstream congestion.

fwRate

The rate, in units of bytes per *ageCoef agingIntervals*, at which fairness eligible traffic is transmitted to the ringlet from the transit queues of the local station.

fwRateCongested

The rate, in units of bytes per *ageCoef agingIntervals*, at which fairness eligible traffic intended for destinations beyond the congestion point, is transmitted to the ringlet from the transit queues of the local station.

localCongested

A boolean value indicating whether or not the local station is congested.

localFairRate

The rate at which the station is allowed to add client-supplied fairness eligible traffic to the ringlet in the absence of downstream congestion, specified in units of bytes per *ageCoef agingIntervals*.

lpAddRate
A smoothed version of the *addRate* obtained by applying the low pass filter function to that variable.

lpAddRateCongested
A smoothed version of the *addRateCongested* obtained by applying the low pass filter function to that variable.

lpFwRate
A smoothed version of the *fwRate* obtained by applying the low pass filter function to that variable.

lpFwRateCongested
A smoothed version of the *fwRateCongested* obtained by applying the low pass filter function to that variable.

lpNrXmitRate
A smoothed version of the *nrXmitRate* obtained by applying the low pass filter function to that variable.

normCoef
The coefficient used to normalize (9.1.9) a local rate value to a rate value that can be uniformly interpreted by all stations on the ringlet. The value of *normCoef* is computed as the product of *ageCoef*, *rateCoef*, and *localWeight* (i.e., *normCoef = ageCoef * rateCoef * localWeight*).

normLocalFairRate
The normalized (9.1.9) value of the *localFairRate*.

normLpFwRate
The normalized (9.1.9) value of *lpFwRate*.

normLpFwRateCongested
The normalized (9.1.9) value of *lpFwRateCongested*.

nrXmitRate
The rate, in units of bytes per *ageCoef agingIntervals*, at which unreserved (i.e., not included in *rateA0*) traffic is transmitted to the ringlet from the add queues and transit queues of the local station.

rateCoef
The coefficient used to scale a locally computed rate to a standard LINK_RATE of 2.5Gbps. The rateCoef is assigned a value of 1 for values of LINK_RATE between 0 and 2.5 Gbps. (inclusive) and a value of 4 for values of LINK_RATE between 2.5 Gbps (exclusive) and 10 Gbps (inclusive).

rcvdRate
The value of the *fairRate* field carried by the most recently received SCFF. This value represents the *normLocalFairRate* of the congestion point originating the SCFF or, in the absence of congestion, the reserved value of FULL_RATE.

rcvdRi
The value of the *ri* field carried by the most recently received SCFF. This value represents the ringlet on which the SCFF originated.

rcvdSa
The value of the *sa* field carried by the most recently received SCFF. This value represents the MAC address of the downstream congestion point or, in the absence of congestion, the MAC address of the upstream neighbor.

rcvdTtl
The value of the *ttl* field carried by the most recently received SCFF after it has been decremented by 1 at the receiving station.

receivedFEFrame
An array of type boolean having MAX_STATIONS entries, intended to record whether the local station has received at least one fairness eligible frame from a particular station during the activeWeightsInterval. This information is used to compute active weights at the expiration of the activeWeightsInterval.

reportingInterval
The time between the sending of MCFFs. The value of the *reportingInterval* is *reportCoef advertisingIntervals* (i.e., *reportingInterval = reportCoef * advertisingInterval*).

sizeFF

The size of a fairness frame as specified in 8.4.

unreservedRate

The rate available for transmission on the ringlet after *rateA0* has been excluded (i.e., *unreservedRate* = LINK_RATE - *rateA0*). The *unreservedRate* is specified in units of bytes per *ageCoef agingIntervals*.

9.3.7 Functions

ActiveWeightsIntervalExpired

Indicates whether or not the activeWeightsInterval has expired.

TRUE: The activeWeightsInterval has expired.

FALSE: The activeWeightsInterval has not expired.

AddByte

Indicates whether a byte of a locally sourced frame is available for per-byte processing.

Values:

TRUE: A byte is available.

FALSE: A byte is not available.

AdvertisingIntervalExpired

Indicates whether the current *advertisingInterval* has expired.

Values:

TRUE: The *advertisingInterval* has expired.

FALSE: The *advertisingInterval* has not expired.

AgingIntervalExpired

Indicates whether the current *agingInterval* has expired.

Values:

TRUE: The *agingInterval* has expired.

FALSE: The *agingInterval* has not expired.

ComputeActiveWeights

for (*activeWeights* = 0, *station* = 0; *station* < *numStations*; *station*++)

if (*receivedFeFrame*[*station*]) *activeWeights* += weights[station];

DiscardFrame

Discard the current fairness frame.

EntryInStq

Provided by Clause 6 to indicate whether there is at least one full frame in the STQ. Allowed values are {TRUE; FALSE}.

FrameReceived

To indicate if a fairness frame has been received by the MAC and is available for processing by the fairness procedures.

Values:

TRUE: A frame has been received from the MAC datapath.

FALSE: A frame has not been received from the MAC datapath.

FrttExpired

Indicates whether the FRTT has expired.

TRUE: The FRTT has expired.

FALSE: The FRTT has not expired.

FwByte

Indicates whether a byte of a transit frame is available for per-byte processing.

Values:

TRUE: A byte is available.

FALSE: A byte is not available.

IsCongested()

Evaluate local congestion conditions.

{

```
        if dualQueueMAC && (stqDepth > stqLowThreshold)
            return TRUE;
        else if !dualQueueMAC &&
            (lpNrXmitRate > rateLowThreshold)
            || classBAccessDelayTimerExpired
            || classCAccessDelayTimerExpired
            return TRUE;
        else
            return FALSE;
```

ReportingIntervalExpired

TRUE: The reportingInterval has expired.

FALSE: The reportingInterval has not expired.

ResetAdvertisingTimer

Set/Reset the *advertisingInterval* timer to expire at the end of the next *advertisingInterval*.

ResetActiveWeightsIntervalTimer

Set/Reset the *activeWeightsInterval* timer to expire at the end of the next *activeWeightsInterval*.

ResetAgingTimer

Set/Reset the *agingInterval* timer to expire at the end of the next *agingInterval*.

ResetFrttTimer

Set/Reset the Frtt timer to expire at the end of the next FRTT.

ResetReportingTimer

Set/Reset the *reportingInterval* timer to expire at the end of the next *reportingInterval*.

SetAllowedRateCongested

```
        if (rcvdRate != FULL_RATE)
            allowedRateCongested = rcvdRate * normCoef;
        else
            allowedRateCongested += (maxAllowedRate - allowedRateCongested) / rampCoef;
```

TransmitFrame

Send the current fairness frame.

9.3.8 Per-byte activities state table

The per-byte activities are performed for each byte of data that transits the station or becomes available for addition to the ringlet. These activities include the increment of rate counters (9.1.3.3) and the setting of policing indications (9.1.3.4). It is permissible to delay the processing of bytes in order to allow as many as 256 bytes to be processed at one time.

Row 9.2-1: The station is in the START state and initializes rate counters and policing indicators. The station makes a transition to the TEST state.

Row 9.2-2: The station is in the TEST state. A byte of a locally-sourced fairness eligible frame is received for addition to the ringlet, bound for a destination beyond the congestion point. The *addRate* and the *addRateCongested* are incremented The station makes a transition to the THRU state.

Row 9.2-3: The station is in the TEST state. A byte of an LSFE frame is received for addition to the ringlet. The *addRate* is incremented. The station makes a transition to the THRU state.

Row 9.2-4: The station is in the TEST state. A byte of a transiting fairness eligible frame is dequed, bound for a destination beyond the congestion point. The *fwRate* and the *fwRateCongested* are incremented The station makes a transition to the THRU state.

Row 9.2-5: The station is in the TEST state. A byte of a transiting fairness eligible frame is dequed. The *fwRate* is incremented. The station makes a transition to the THRU state.

Row 9.2-6: The station is in the TEST state. A byte meeting none of the previous conditions of the TEST state is added or transited. No action is taken but the station makes a transition the THRU state.

Row 9.2-7: The station is in the THRU state. The added or transited byte is not of service classA0 (i.e., is unreserved). The nrXmitRate is incremented. The station makes a transition to the sets state.

Row 9.2-8: The station is in the THRU state., the added or transited byte does not meet any listed conditions. The station makes a transition to the SETS state.

Table 9.2—Per-byte activities

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

START	—	1	addRate = 0;
			addRateCongested = 0;
			fwRate = 0;
			fwRateCongested = 0;
			nrXmitRate = 0;
			addRateOK = TRUE;
			addRateCongestedOK = TRUE;
				TEST

TEST	AddByte() && ((frame.sc == classB) && frame.fe) || (frame.sc == classC)) && SentBeyondCongestionPoint()	2	addRate += 1; addRateCongested += 1;	THRU
	AddByte() && ((frame.sc == classB) && frame.fe) || (frame.sc == classC))	3	addRate += 1;	
	FwByte() && ((frame.sc == classB) && frame.fe) || (frame.sc == classC)) && SentBeyondCongestionPoint()	4	fwRate += 1; fwRateCongested += 1;	
	FwByte() && ((frame.sc == classB) && frame.fe) || (frame.sc == classC))	5	fwRate += 1;	
	AddByte() || FwByte()	6	—	
THRU	(AddByte() || FwByte()) && frame.sc != CLASS_A0	7	nrXmitRate+=1;	SETS
	—	8	—	
SETS	—	9	addRateOK = addRate < allowedRate && nrXmitRate < unreservedRate && (!EntryInStq() || (fwRate > addRate && stqDepth < stqHighThreshold)); addRateCongestedOK = addRateOK && (addRateCongested < allowedRateCongested);	TEST

Row 9.2-9: The station is in the SETS state. The *addRateOK* and the *addRateCongestedOK* policing parameters are set as described in 9.1.3.4. The station makes a transition to the TEST state to process the next added or transited byte.

9.3.9 Per-agingInterval state table

The per-agingInterval state table performs low pass filtering of rate counters, normalization of rates, and aging of rate counters. Aging and low pass filtering are described in 9.1.3.3. Normalization is described in 9.1.3.2.

Table 9.3—Common agingInterval updates

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

START	—	1	lpAddRate = 0;
			lpAddRateCongested =0;
			lpFwRate = 0;
			lpFwRateCongested = 0;
			lpNrXmitRate = 0;
			normLpFwRate = 0;
			normLpFwRateCongested = 0;
				WAIT

WAIT	AgingIntervalExpired()	2	lpaddRate += (addRate - lpAddRate) / lpCoef;
			lpaddRateCongested +=
			(addRateCongested - lpAddRateCongested) / lpCoef;
			lpfwRate += (fwRate - lpFwRate) / lpCoef;
			lpfwRateCongested +=
			(fwRateCongested - lpFwRateCongested) / lpCoef;
			lpNrXmitRate += (nrXmitRate - lpNrXmitRate) / lpCoef;
			normLpFwRate = lpFwRate / normCoef;
			normLpFwRateCongested =
			lpFwRateCongested / normCoef;
			addRate -= addRate / ageCoef;
			addRateCongested -= addRateCongested / ageCoef;
			fwRate -= fwRate / ageCoef;
			fwRateCongested -= fwRateCongested / ageCoef;
			nrXmitRate -= nrXmitRate / ageCoef;
			agingUpdateDone = TRUE;
			ResetAgingTimer();
				WAIT

	—	3	—	

Row 9.3-1: The station is in the START state and initializes low pass filtered rates and normalized rates. The station makes a transition to the WAIT state.
Row 9.13-2: The station is in the WAIT state and an *agingInterval* has expired. low pass filtering (9.1.3.3) of rate counters, normalization (9.1.3.2) of smoothed rates, and aging of rate counters is performed. An flag is set indicating that the aging operation has been completed and the adjustment-method specific portion of agingInterval processing can proceed. The agingInterval timer is reset to start the next agingInterval and the station remains in the WAIT state.
Row 9.13-3: The station is in the WAIT state and the *agingInterval* has not expired. The table remains in the WAIT state.

9.3.10 Aggressive rate adjustment state table

At the completion of *agingInterval* common processing, the indication *agingIntervalDone*, allows the start of activities associated with aggressive rate computation. These activities include the computation of the localFairRate, normLocalFairRate, allowedRate, allowedRateCongested, and hopsToCongestion. Allowed-rate information is optionally supplied to the MAC client.

The following activities are performed by all rows of the state table except the first row (START state). They are not repeated in the individual row descriptions following the state table:

a) The *allowedRateCongested* is set as described by (9.1.3.7.).

b) The variable *localCongested* is set appropriately to indicate whether or not the station is locally congested. This variable is referenced by other state tables. The conditions associated with local congestion are described in 9.1.3.
c) The *normLocalFairRate* is computed if a new value of *localFairRate* has been assigned. Normalization is described in 9.1.3.2.
d) An MA_CONTROL.indication is optionally issued to the MAC client with allowed-rate information (5.3).

The condition *AgingUpdateDone* is checked in all rows of the state table except the first row (START state) but does not appear in the individual row descriptions below the table..

Table 9.4—Aggressive rate adjustment

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

START	—	1	localCongested = FALSE;
			allowedRate = maxAllowedRate;
			localFairRate = unreservedRate;
				UNCG

UNCG	agingUpdateDone && IsCongested()	2	localCongested = TRUE;
			localFairRate = lpAddRate;
			normLocalFairRate = localFairRate / normCoef;
			SetAllowedRateCongested (rcvdRate);
			MA_CONTROL.indication(SINGLE_CHOKE_IND);
			agingUpdateDone = FALSE;
				CGST

	agingUpdateDone	3	SetAllowedRateCongested(rcvdRate); MA_CONTROL.indication(SINGLE_CHOKE_IND); agingUpdateDone = FALSE;	UNCG

CGST	agingUpdateDone && !IsCongested()	4	localCongested = FALSE;
			localFairRate = unreservedRate;
			normLocalFairRate = localFairRate / normCoef;
			SetAllowedRateCongested(rcvdRate);
			MA_CONTROL.indication(SINGLE_CHOKE_IND);
			agingUpdateDone = FALSE;
				UNCG

	agingUpdateDone	5	localFairRate = lpAddRate;
			normLocalFairRate = localFairRate / normCoef;
			SetAllowedRateCongested(rcvdRate);
			MA_CONTROL.indication(SINGLE_CHOKE_IND);
			agingUpdateDone = FALSE;
				CGST

Row 9.4-1: The station is in the START state and variables are initialized. The *allowedRate* does not change when using the aggressive method and can be initialized to the value of *maxAllowedRate* (9.1.3.6). A transition is made to the UNCG (uncongested) state.
Row 9.4-2: The station is in the UNCG state and detects congestion (9.1.3). The *localFairRate* is set to the smoothed LSFE rate of addition to the ringlet (*lpAddRate*). A transition is made to the CGST (congested) state.
Row 9.4-3: The station is in the UNCG state and does not detect congestion (9.1.16). The *localFairRate* is unchanged and there is no state transition.
Row 9.4-4: The station is in the CGST state and does not detect congestion (9.1.16). The *localFairRate* is set to the *unreservedRate* (i.e., LINK_RATE - *rateA0*). A transition is made to the UNCG state.
Row 9.4-5: The station is in the CGST state and detects local congestion (9.1.16). The *localFairRate* is set to the smoothed LSFE rate of addition to the ringlet (*lpAddRate*). No state transition is made.

9.3.11 Conservative rate adjustment state table

At the completion of *agingInterval* common processing, the indication *agingIntervalDone*, allows the start of activities associated with conservative rate computation. These activities include the computation of the localFairRate, normLocalFairRate, allowedRate, allowedRateCongested, and hopsToCongestion. Allowed-rate information is optionally supplied to the MAC client..

The following activities are performed by all rows of the state table except the first row (START state). They are not repeated in the row descriptions that follow the state table:

a) The *allowedRate* is assigned a value according to the next-state associated with the row. If the next state is the UNCG state, the *allowedRate* is ramped up. If the next state is the CGST state, the *allowedRate* is assigned the value of the smaller of the *unreservedRate* and the *localFairRate* (9.1.3.6).
b) The *allowedRateCongested* is assigned a value depending on whether or not downstream congestion is detected. If downstream congestion is detected, the *allowedRateCongested* is assigned the weight-adjusted value of the *localFairRate* associated with the downstream congestion point. If the station is not downstream congested, the *allowedRateCongested* is ramped up (9.1.3.7.)
c) An MA_CONTROL.indication is optionally issued to the MAC client with allowed-rate information (5.3).
d) A variable (*localCongested*) is set to reflect the current congestion (9.1.3) condition. This variable is referenced in other state tables.
e) The *normLocalFairRate* is computed and the FRTT timer is reset if a new value of *localFairRate* has been assigned.

Also, the condition *agingIntervalDone* occurs in all rows of the state table except the first row (START state) and the description is not repeated in the row descriptions that follow the state table.

Row 9.5-1: The station is in the START state. Variables are initialized and a transition is made to the UNCG state.
Row 9.5-2: The station is in the UNCG state and local congestion is detected (9.1.16). If the station implements *activeWeightsDetection*, the initial value of the *localFairRate* is calculated as a fair share of the unreservedRate. A fair share is given by the ratio of the *localWeight* to the *activeWeights* (see *ActiveWeightsComputation())*. If the station does not implement *activeWeightsDetection*, the *localFairRate* is set to the smoothed LSFE rate of addition to the ringlet (*lpAddRate*). In either case, the value assigned serves as an initial value of the *localFairRate* and changes during the period of congestion. A transition is made to the CGST state.
Row 9.5-3: The station is in the UNCG state. Local congestion is not detected. No state transition occurs. The localFairRate is unchanged and the allowedRate is ramped up.
Row 9.5-4: The station is in the CGST state. Hysteresis is applied to the transition between CGST and UNCG states by requiring that the normalized *localFairRate* (*normlocalFairRate)* increase to a value close to the *unreservedRate* before the transition to the UNCG state is made.
Row 9.5-5: The station is in the CGST state. STQ occupancy (in the dual-queue case) or rate (in the single-queue case) grows beyond a specified threshold value and a fairness round trip time has elapsed since the last change in the value of the localFairRate. The localFairRate is ramped down but is not reduced below the station's fair share of the unreserved rate. No state transition is made.
Row9.5-6: The station is in the CGST state. STQ occupancy (in the dual-queue case) or rate (in the single-queue case) falls below a specified threshold value and a fairness round trip time has elapsed since the last change in the value of the localFairRate. The localFairRate is ramped up. No state transition occurs.
Row 9.5-7: The station is in the CGST state. The station determines that severe congestion exists and as a result it is adding less than its fair share. The station recomputes the *localFairRate* to get a more current estimate. No state transition occurs.

Table 9.5—Conservative rate adjustment

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	—	1	localCongested = FALSE; localFairRate = unreservedRate;	UNCG

UNCG	agingUpdateDone && IsCongested()	2	if activeWeightsDetection
			  localFairRate =
			  (unreservedRate / activeWeights) * localWeight;
			else
			  localFairRate = lpAddRate;
			normLocalFairRate = localFairRate / normCoef;
			localCongested = TRUE;
			allowedRate = min(unreservedRate, localFairRate);
			ResetFrttTimer();
			SetAllowedRateCongested(rcvdRate);
			MA_CONTROL.indication(SINGLE_CHOKE_IND);
			agingUpdateDone = FALSE;
				CGST

	agingUpdateDone	3	AllowedRate += (maxAllowedRate - allowedRate) / rampCoef; SetAllowedRateCongested (rcvdRate); MA_CONTROL.indication(SINGLE_CHOKE_IND); agingUpdateDone = FALSE;	UNCG

Table 9.5—Conservative rate adjustment *(continued)*

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

CGST	agingUpdateDone && ((localFairRate / localWeight) >= unreservedRate - rampCoef)	4	localCongested = FALSE;
			allowedRate +=
			(maxAllowedRate - allowedRate) / rampCoef;
			SetAllowedRateCongested (rcvdRate);
			MA_CONTROL.indication(SINGLE_CHOKE_IND);
			agingUpdateDone = FALSE;
				UNCG

	agingUpdateDone && ((dualQueueMAC && (stqDepth > stqMedThreshold)) || (!dualQueueMAC && (addRate+fwdRate > rateHighThreshold))) && FrttTimerExpired()	5	if activeWeightsDetection
			localFairRateLowerBound =
			(lpaddRate + lpfwRate) * (localWeight / activeWeights);
			else
			localFairRatelowerbound = lpAddRate;
			localFairRate = max(localFairRateLowerBound,
			localFairRate - localFairRate / rampCoef);
			normLocalFairRate = localFairRate / normCoef;
			allowedRate = min(unreservedRate, localFairRate);
			ResetFrttTimer();
			SetAllowedRateCongested(rcvdRate);
			MA_CONTROL.indication(SINGLE_CHOKE_IND);
			agingUpdateDone = FALSE;
				CGST

	agingUpdateDone && ((dualQueueMAC && (stqDepth < stqLowThreshold)) || !dualQueueMAC && (addRate + fwRate < rateLowThreshold)) && FrttTimerExpired()	6	localFairRate += (localWeight * (unreservedRate -
			lpAddRate - lpFwRate)) / rampCoef;
			normLocalFairRate = localFairRate / normCoef;
			allowedRate = min(unreservedRate, localFairRate);
			ResetFrttTimer();
			SetAllowedRateCongested (rcvdRate);
			MA_CONTROL.indication(SINGLE_CHOKE_IND);
			agingUpdateDone = FALSE;	

	agingUpdateDone	7	if (stqDepth() > stqHighThreshold) &&
			((activeWeightsDetection &&
			(lpAddRate / localWeight <
			(lpFwRate / (activeWeights - localWeight)))) ||
			(!activeWeightsDetection &&
			lpAddRate < starveFactor * allowedRate))
			{
			if activeWeightsDetection
			localFairRateUpperBound =
			(lpAddRate+lpFwRate)*(localWeight /activeWeights);
			}
			else
			{
			localFairRateUpperBound = lpAddRate;
			localFairRate =
			min(localFairRate, localFairRateUpperBound);
			}
			normLocalFairRate = localFairRate / normCoef;
			allowedRate = min(unreservedRate, localFairRate);
			SetAllowedRateCongested (rcvdRate);
			MA_CONTROL.indication(SINGLE_CHOKE_IND);
			agingUpdateDone = FALSE;	

9.3.12 Per-advertisingInterval state table

At the expiration of each *advertisingInterval* the local station sends an *advertisedFairRate* to its upstream neighbor. The computation of the *advertisedFairRate* reflects the rule that a station advertises the *normLocalFairRate* associated with the head of a congestion domain in which it lies, unless it is the tail of that domain. A station that does not lie within a congestion domain, or is the tail of a downstream congestion domain but not the head of an upstream congestion domain, advertises the value FULL_RATE.

Table 9.6—Per-advertisingInterval

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

START	—	1	rcvdRate = FULL_RATE;
			rcvdRi = myRingletID;
			rcvdSa = myMACAddress;
			rcvdTTL = MAX_STATIONS;
			ResetAdvertisingTimer();
				WAIT

WAIT	AdvertisingIntervalExpired() &&
	localCongested &&
	(!downstreamCongested ||
	(normLocalFairRate ≤ rcvdRate))
		2	frame.ffType = SINGLE_CHOKE;
			frame.fairRate = normLocalFairRate;
			frame.sa= myMACAddress;
			frame.ttl = MAX_STATIONS;
			frame.ri = myRingletID;
			TransmitFrame();
			ResetAdvertisingTimer();
				WAIT

	AdvertisingtIntervalExpired() &&
	downstreamCongested &&
	(!localCongested ||
	(normLocalFairRate > rcvdRate))
		3	frame.ffType = SINGLE_CHOKE;
			if (ringTopologyType == RING_CLOSED)
			&& (rcvdRate >=
			localWeight * normLpFwRateCongested)) ||
			(ringTopologyType == RING_OPEN &&
			rcvdRi != myRingletID &&
			(rcvdRate ≥ localWeight * normLpFwRate))
			{
			frame.fairRate = FULL_RATE;
			frame.sa = myMACAddress;
			frame.ttl = MAX_STATIONS;
			frame.ri = myRingletID;
			}
			else
			{
			frame.fairRate = rcvdRate;
			frame.sa = rcvdSa;
			frame.ttl = rcvdTtl;
			frame.ri = rcvdRi;
			}
			TransmitFrame();
			ResetAdvertisingTimer();	

	AdvertisingIntervalExpired() &&
	!downstreamCongested &&
	!localCongested
		4	frame.ffType = SINGLE_CHOKE;
			frame.ttl = MAX_STATIONS;
			frame.ri = myRingletID;
			frame.fairRate = FULL_RATE;
			frame.sa = myMACAddress;
			TransmitFrame();
			ResetAdvertisingTimer();	

Row 9.6-1: The station is in the START state. Variables are initialized and a transition is made to the WAIT state.

Row 9.6-2: The station is in the WAIT state. The local station is more congested than any downstream station or there is no downstream congestion. The *normLocalFairRate* is advertised. No state transition is made.

Row 9.6-3: The station is in the WAIT state. A downstream station is more congested than the local station.

The *rcvdRate* (*normlocalFairRate* of the congestion point) is advertised. An exception is made if it is determined that no single upstream station could be sending at a rate that is more restricted than that of the local station. In this case, the congestion can be terminated at the local station by advertising the value FULL_RATE. No state transition is made.
Row 9.6-4: The station is in the WAIT state. Neither downstream nor local congestion is detected. The FULL_RATE is advertised. No state transition is made.

9.3.13 Per-reportingInterval state table

At the expiration of a *reportingInterval*, the local station broadcasts an MCFF to all stations on the ringlet. Per-*reportingInterval* processing is further explained in 9.1.25.

Table 9.7—Per-reportingInterval

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	—	1	ResetReportingTimer();	WAIT

WAIT	ReportingIntervalExpired() && localCongested	2	frame.ffType = MULTI_CHOKE;
			frame.fairRate = normLocalFairRate;
			frame.sa= myMACAddress;
			frame.ttl = MAX_STATIONS;
			frame.ri = myRingletID;
			TransmitFrame();
			MA_CONTROL.indication (MULTI_CHOKE_IND);
			ResetReportingTimer();
				WAIT

	ReportingIntervalExpired()	3	frame.ffType = MULTI_CHOKE;
			frame.fairRate = FULL_RATE;
			frame.sa= myMACAddress;
			frame.ttl = MAX_STATIONS;
			frame.ri = myRingletID;
			TransmitFrame();
			MA_CONTROL.indication (MULTI_CHOKE_IND);
			ResetReportingTimer();	

Row 9.7-1: The station is in the START state. A transition is made to the WAIT state.
Row 9.7-2: The station is in the WAIT state. The local station is congested. Report the *normLocalFairRate* to all stations on the ringlet via MCFF.
Row 9.7-3: The local station is not congested. Report the FULL_RATE to all stations on the ringlet via MCFF.

9.3.14 Per-activeWeightsInterval state table

This state table is applicable only when using conservative rate adjustment and when *activeWeightsDetection* is TRUE.

At the expiration of an *activeWeightsInterval*, the local station computes the sum of the weights of stations from which a fairness eligible frame has been received during the past *activeWeightsInterval*. The *activeWeightsInterval* is configured as an integer multiple of the *agingInterval*. The *activeWeights* value is referenced during the per-*agingInterval* activities.

Table 9.8—Per-activeWeightsInterval

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	—	1	activeWeights = 1; ResetActiveWeightsIntervalTimer();	WAIT
WAIT	activeWeightsIntervalExpired();	2	computeActiveWeights(); ResetActiveWeightsIntervalTimer();	WAIT

Row 9.8-1: The station is in the START state. Variables are initialized and a transition is made to the WAIT state.
Row 9.8-2: The station is in the WAIT state. An *activeWeightsInterval* has expired. An *activeWeights* computation is performed. No state transition is made.

9.3.15 Receive fairness frame state table

Fairness frames are processed as they arrive. Frames are validated and identified by type. The information in an SCFF is saved after the received *ttl* is decremented. The multichoke information associated with an MCFF is optionally transferred to the MAC client. Fairness message receive processing is further explained in 9.1.26.

Row 9.9-1: The station is in the START state. Variables are initialized and a transition is made to the WAIT state.
Row 9.9-2: The station is in the WAIT state. A frame is received with an expired time to live. The fairness frame is discarded and the station remains in the WAIT state.
Row 9.9-3: The station is in the WAIT state. An SCFF is received. The station remains in the WAIT state.
Row 9.9-4: MCFF received. The MC_FF is discarded if the local fairness instance is the origin of the MC_FF. MA_CONTROL.indication (see Table 5.3) is optionally issued to the client to indicate that multi choke information is available. The station remains in the WAIT state.

9.4 Fairness MIB attributes

9.5 Fairness frame format

The fairness frame format is specified by 8.4 and illustrated by Figure 8.8. The values assigned to the *baseRingControl* sub-fields of a fairness frame are specified by Table 9.11.

The *ttl* (time to live) field is set to the value MAX_STATIONS by an originating station. Each subsequent station in a congestion domain sets the *ttl* to one less than the *ttl* of the last received SCFF. ThisThis allows a receiving station to compute the number of hops to the originating station as MAX_STATIONS - *frame.ttl*. A station is the origin of a fairness frame if it placed *myMACAddress* in the *frame.sa* field of the transmitted frame.

The fairness frame payload contains a 16-bit *fairnessHeader* and a 16-bit *fairRate* as illustrated by Figure 9.12.

Individual fields of the fairness frame payload are described in the sections that follow.

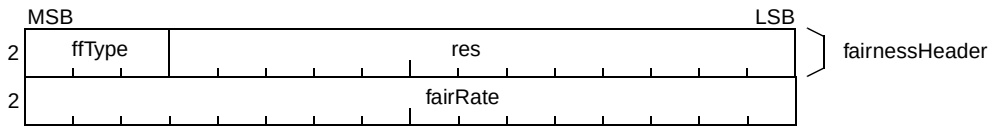

Figure 9.12—Fairness frame payload

Table 9.9—Receive fairness frame

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	—	1	—	WAIT
WAIT	FrameReceived() && frame.ttl == 0	2	Discard();	WAIT

	FrameReceived() &&
	frame.ft == FT_FAIRNESS
	&&
	frame.ffType == SC_FF
		3	rcvdRate = frame.fairRate;
			rcvdSa = frame.sa;
			rcvdTtl = frame.ttl - 1;
			rcvdRi = frame.ri;
			if (frame.sa == myMACAddress) &&
			((frame.ri == myRingletID) || (EdgeOfRing() &&
			myWrappingMethod == CENTER_WRAP))
			{
			rcvdRate = FULL_RATE;
			hopsToCongestion = MAX_STATIONS;
			}
			if (rcvdRate != FULL_RATE)
			{
			downstreamCongested = TRUE;
			if (ringTopologyType == RING_OPEN) &&
			(rcvdRi != myRingletID)
			hopsToCongestion = hopsToDsWrap;
			else
			hopsToCongestion = MAX_STATIONS - rcvdTtl;
			}
			else
			{
			downstreamCongested = FALSE;
			}
			Discard();	

	FrameReceived() &&
	frame.ft == FAIRNESS) &&
	frame.ffType == MC_FF)
		4	if (frame.sa == myMACAddress) &&
			((frame.ri == myRingletID) || (EdgeOfRing() &&
			(myWrappingMethod == CENTER_WRAP)))
			{
			Discard();
			}
			else
			{
			MA_CONTROL.indication(MULTI_CHOKE_IND);
			Discard();
			}	

Table 9.10—Fairness MIB attributes

MIB attribute name	Row	Definition	Variable name
TBD	1	TBD	*advertisementRatio*
TBD	2	TBD	*ageCoef*
TBD	3	TBD	*checkAccessDelayTimer*
TBD	4	TBD	*checkRateThreshold*
TBD	5	TBD	*localWeight*
TBD	6	TBD	*lpCoef*
TBD	7	TBD	*maxAllowedRate*
TBD	8	TBD	*rampCoef*
TBD	9	TBD	*rateHighThreshold*
TBD	10	TBD	*rateLowThreshold*
TBD	11	TBD	reportCoef
TBD	12	TBD	*stqFullThreshold*
TBD	13	TBD	*stqHighThreshold*
TBD	14	TBD	*stqLowThreshold*
TBD	15	TBD	*stqMedThreshold*
TBD	16	TBD	*weightComputationCoef*

Table 9.11—Fairness frame baseRingControl values

Field	Value
ft (frame type)	FT_FAIRNESS
sc (service class)	CLASS_A0
fe (fairness eligible)	FALSE
we (wrap eligible)	TRUE
ri (ringlet identifier)	~~!myRingletID~~
parity	(see 8.4.2.6)

9.5.1 ffType

Interpretations of the 3-bit *ffType* field are specified by Table 9.12.

9.5.2 res

The 13-bit *res* field is ignored on receipt and set to zero on transmit.

Table 9.12—Fairness frame type (ffType) values

Value	Name	Usage
000_2	SINGLE_CHOKE	Provides the advertisedFairRate of a station to the upstream neighbor once per advertisementInterval.
001_2	MULTI_CHOKE	Provides the normLocalFairRate of a station to all other stations on the ringlet once per reporting interval.
010_2 - 111_2	reserved	For future use.

9.5.3 fairRate

The 16-bit *fairRate* field carries a normalized rate encoded as a 16-bit quantity. A value of FULL_RATE ($FFFF_{16}$) indicates the full line rate.

9.6 Explanation of ranges and default values (informative)

Editors' Notes: *To be removed prior to final publication.*
This section addresses numerous comments (e.g., Jan '03 #831, #842, #846, #890) requesting explanations as to how default values and allowed ranges for configured variables (9.3.5) were selected. It is expected that the explanation will take the form of a numeric analysis or reference to a specific simulation study.

9.7 Effects of lost fairness frames (informative)

Editors' Notes: *To be removed prior to final publication.*
This section needs editing and clarification. The editor is aware of this.

The *allowedRateCongested* maintains its current value until the next SCFF is received. Therefore, the loss of an SCFF would result in one of the following. If the lost frame contained a new value of FULL_RATE, it would prevent the station from ramping up its allowed rate for the ringlet. If the value contained in the lost frame was a new non-FULL_RATE value, the MAC will not be able to react by setting its allowed rate to the received advertised rate, which could have been higher or lower than the current allowed rate. If the lost frame did not contain a new value from the previous advertisement, there will be no effect on the fairness algorithm.

Editors' Notes: *To be removed prior to final publication.*
The following statement has been modified (comment #202 11/02) and will be updated when the text described below (from Jason Fan) is available in Clause 11.

Refer to Clause 11 for the protection implications of not receiving consecutive SCFFs.

The loss of an MCFF may result in the failure of one or more clients to learn the current value of the *advertisedFairRate* of the station originating the frame. The client and the MAC shapers will not be synchronized. This may cause the client to attempt to send more or less than it actually should.

9.8 Explanation of aging and rates (informative)

It was described in 9.1.3.3 that the rate counters *addRate*, *addRateCongested*, *fwRate*, *fwRateCongested*, and *nrXmitRate* are conditionally incremented with each byte of a data frame that is dequeued by the datapath

for addition to the ringlet or transited by the station. There are a number of ways that a rate counter could provide a rate value. One method would be to allow the rate counter to maintain a simple count over time. The rate could be derived by dividing the count by the number of time intervals that have elapsed. This method has the disadvantage that the counter will overflow, as there is no provision for reducing the count. An alternative would be to store the counts associated with the previous N intervals. When an interval is complete, the new count is recorded, and the oldest count is dropped. While this method avoids the overflow of the rate counter, it also introduces the requirement to store N counts. The aging method employed by the fairness procedures ensures that the rate counter will not overflow and does not require more than a single count to be maintained for any individual rate counter.

The aging method has three objectives:

a) Ensuring that the rate counters do not overflow.
b) Smoothing the rate counter values.
c) Transforming the rate counter values to rates.

Taking *addRate* as an example, the rate counter is aged as follows at the expiration of each *agingInterval*:

```
addRate = (addRate * (ageCoef - 1)) / ageCoef;
```

That is, at each *agingInterval*, the *addRate* is incremented by the number of added bytes, and decremented by (1 / *ageCoef*).

As an illustration, assume that *ageCoef* is 4, that *addRate* is initially 0, and that the *agingInterval* is 100 usec (1 / 10**4 seconds). Assume also that bytes are added at a constant rate of 10**7 bytes/second, that is, 1000 bytes are added every *agingInterval*.

During the first aging interval, *addRate* is incremented from 0 to 1000, but aging at the end of the interval sets it back to 1000 * 3/4 = 750.

The next interval increments the counter to 1750. Aging sets it back to 1750 * 3/4=1312.

The next 1000 bytes increases the counter to 2312. Aging sets it back to 1734.

The next 1000 bytes increases the counter to 2734. In subsequent *agingIntervals*, the count takes the values (at the end of the aging interval, but before aging) 3050, 3287, 3465, 3598, 3698, 3773, etc.. The first 40 values are illustrated in Figure 9.13.

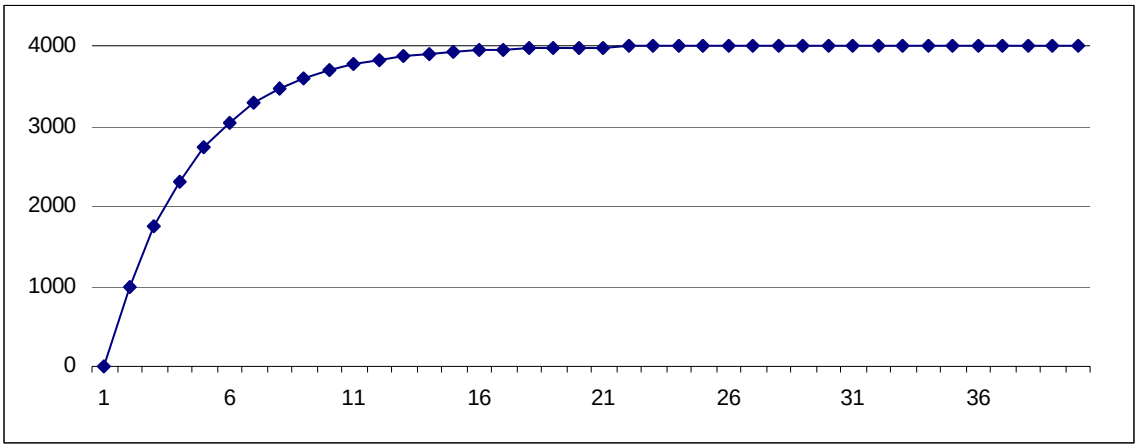

Figure 9.13—rate counter asymptotically approaching rate

The value of *addRate* asymptotically approaches 4000. If it becomes 4000, aging decreases it to 4000 * 3/4 = 3000. At the addition of 1000 bytes at the next *agingInterval*, the addRate again reaches 4000. It can be seen that the *addRate* will not exceed 4000 (i.e., 1000 * ageCoef).

It can be observed also that the value of the *addRate* is *ageCoef* times the actual rate of byte addition in units of bytes per *agingInterval*. It follows that the unit associated with the *addRate* (and rate counters in general) is bytes per *ageCoef agingIntervals*. Many rates used in the fairness procedures are derived from the rate counters, so it follows that this unit is commonly used in the fairness procedures.

It follows also that the value of a rate counter can be converted to the more conventional unit of bytes per second as follows:

$$\text{rate}_{\text{bytes/sec}} = \text{addRate} / (\text{ageCoef} * \text{agingIntervals}) \quad (9.1)$$

Using the values from the example above:

$$\text{rate} = 4000_{\text{bytes per ageCoef agingIntervals}} / (4 * 10^{-4}{}_{\text{seconds per agingInterval}}) = 10^{7}{}_{\text{bytes/sec}} \quad (9.2)$$

9.9 Protocol Implementation Conformance Statement (PICS) proforma for Clause 9[1]

Editors' Notes: *To be removed prior to final publication.*

This PICS proforma is at present incomplete. No valid items should be assumed to be present in this proforma. Conformance of any type to any portion of this clause shall not be claimed on the basis of any item in this section.

This standards draft shall not be considered to be complete until this PICS proforma is complete.

The editors estimate that the level of completeness of this PICS proforma is 0%.

May03 #636 (637, 638, 640): Populate PICS.

9.9.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 9, Fairness, shall complete the following Protocol Implementation Conformation Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-1998.

9.9.2 Identification

9.9.2.1 Implementation identification

Supplier[a]	
Contact point for enquiries about the PICS[a]	
Implementation Name(s) and Version(s)[a,c]	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s))[b]	

a—Required for all implementations

b—May be completed as appropriate in meeting the requirements for the identification.

c—The terms Name and Version should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

[1]*Copyright release for PICS proformas:* Users of this standard may freely reproduce the PICS proforma in this annex so that it can be used for its intended purpose and may further publish the completed PICS.

9.9.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-200x, Resilient Packet Ring Access Method and Physical Layer Specifications, Fairness Control
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [ ] Yes [ ] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-200x.)	

Date of Statement	

9.9.3 Major capabilities/options

Editors' Notes: *To be removed prior to final publication.*
May03 #636 (637, 638, 640): Populate PICS.

Item	Feature	Subclause	Value/Comment	Status	Support
FSM1	Conformance to fairness state machines	9.3	<capability> supported.	M	Yes [ ]
<opt>	<Optional capability>	<xref>	<option> supported.	O	Yes [ ] No [ ]
MD	Management Data Interface	12.1.1.3	Management data interface supported.	O	Yes [ ] No [ ]

9.9.4 PICS tables for Clause 9