dvjClause09NewFairness13.fm

May 19, 2003 9:50 am

Refined fairness

This contribution addresses the definition and enforcement of weighted fairness, based on concepts described RPR meetings and formally described in [B1]. In this reference, fairness requires each station to pay a *fair cost* for congested-link usage, where *fair cost* is determined by the demand for congested-link bandwidth. Weighted fairness provides each station with an income stream, based on its weight, that is used to purchase *fair cost* resources. In RPR, a *fairRate* value serves this purpose, although this value is inversely related to the link-bandwidth cost.

[B1] *The Fairness and Utility of Pricing Network Resources Using Competitive Markets*[1], Errin W. Fulp[2] and Douglas S. Reeves[3], Department of Computer Science, North Carolina State University.

This proposal refines the features contained within D2.2 and earlier versions of the RPR draft. Areas of attention include:

a) Abstraction. Specifications focus on normalized functions, not implementation details. Implementation choices (such as the precision of a multiplier) are abstracted from the specification.
b) Linear. Soft (linear) indications reduce oscillations associated with hard (full-on/full-off) controllers.
c) Shapers. ClassC traffic rates are limited by shapers, rather than average bandwidth comparisons. Shapers have better short and long term behaviors, like other rate controls in Clause 6.
d) Spatial. Explicit spatial awareness facilitates decoupling of largely isolated fairness domains.
e) Simple. Only two data flows (add and transit of fairness eligible) need be monitored.

Pending thoughts:

a) Resolution. To retail *fairRate* resolution, consider limiting cumulative weights to a 4K maximum.

[1]Available at: http://arqos.csc.ncsu.edu/papers/2000_02_cnj.pdf
[2]Email address: ewfulp@eos.ncsu.edu
[3]Email address: reeves@eos.ncsu.edu

9. Fairness applications

9.1 Fairness overview

9.1.1 Fairness domains

A homogeneous topology with random transfer paths have only one fairness domain, as illustrated in Figure 9.1a. For such topologies, congestion locations are unstable and multiple similar-level congestion locations will be present.

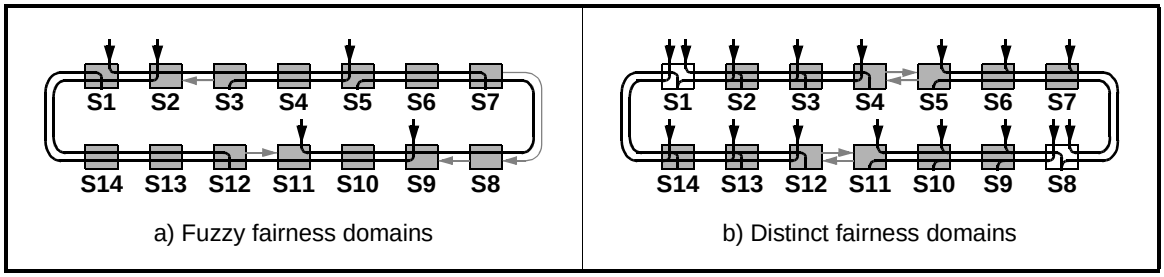


Figure 9.1—Fairness domains

However, topologies with distinct "drain" stations have distinct largely independent fairness domains, as illustrated in Figure 9.1b (the drain stations are shaded white). For such topologies, congestion locations are stable and most traffic only experience drain-station congestion.

Fairness protocols use a single multichoke control mechanism to address both of the (fuzzy and distinct) fairness domain conditions, including:

a) Facilitates spatial awareness of congestion conditions within a single domain.
b) Isolates flow-control dynamics within distinct fairness domains.

9.1.2 Opposing fairness frames

Congestion conditions one ringlet are relieved through fairness control messages sent over the opposing ringlet. Fairness control frames are stripped by each station and updated by the fairness control unit (FCU) before being retransmitted, as illustrated in Figure 9.2a. Congestion of common traffic destined for station S7 is nominally detected at the upstream station S6, whose FCU provides the fairness control messages for upstream stations. The FCUs of less congested upstream stations pass fairness frames, thereby communicating downstream congestion conditions to upstream stations, as conceptualized in Figure 9.2b.

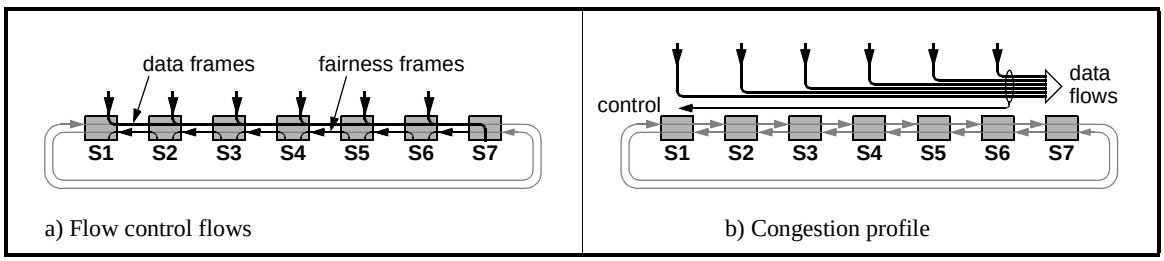


Figure 9.2—Logical data and control flows

9.1.3 Fairness principles

The fairness protocols are influenced by per-station weights w_i and the fraction of choke-point bandwidth consumed by allocated bandwidths, as illustrated in Figure 9.3a. In this illustration, f_a represents the fraction of bandwidth consumed by higher-precedence (classA and classB-CIR) allocated traffic.

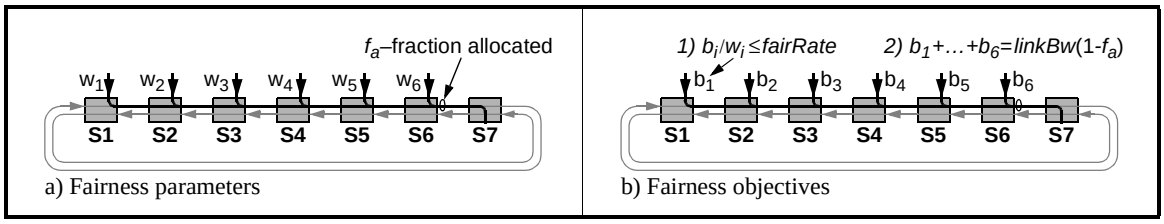


Figure 9.3—Flow controlled stations

The fairness protocols are based on determination of a controlling *fairRate* value, where *fairRate* limits each station's bandwidth consumption to its weighted budget, as illustrated in Figure 9.3b. With the desired value of *fairRate,* the cumulative fairness-eligible bandwidths through choke station S6 equals the non-allocated portion of the that link bandwidth *linkBw.* The *fairRate* factor is inversely proportional to cost, in that smaller values represent the higher costs of highly congested links.

Supporting the fairness objective involves reverse feedback, from the congested S6 station. The congested S6 station constantly communicates the controlling *fairRate* value to upstream stations. That *fairRate* value converges to the desired value through feedback, as described below:

a) When congested S6 traffic is present, *fairRate* adjustments proceed as follows:
   1) $b_6/w_6 < fairRate$: Over-restricted station S6 decreases its asserted *fairRate* value.
      This change and its decrease of upstream bandwidth nudges b_6/w_6 upwards towards *fairRate*.
   2) $b_6/w_6 > fairRate$: Under-restricted station S6 increases its asserted *fairRate* value.
      This change and its increase of upstream-bandwidth nudges b_6/w_6 down towards *fairRate*.
b) When congested traffic is absent, *fairRate* migrates towards the largest (uncongested) value.

The fairness control protocols utilize these basic feedback control strategies, but apply smoothing low-pass filters to measured bandwidths and communicated *fairRate* values. The low-pass filter time constants are selected to achieve fast response times (so that unfair congestion is short lived) while avoiding oscillatory behaviors that could reduce effective fairness-eligible bandwidths.

A complicating factor is the transition from condition (b) to (a). If the duration of (b) processing is short, then (a) processing should be resumed using its previous *fairRate* value. If the duration of (b) is long, then starting with a revised *fairRate* estimate is appropriate. This *fairRate* estimate could be based on the burst bandwidth predictions (based on previous station S6 behaviors) and/or measured upstream traffic profiles.

9.1.4 Congestion controls

Fairness rates are generated by monitoring the fair add and transit bytes, ***addByte*** and ***fwByte***, as illustrated in Figure 9.4. While the specification is based on individual byte counts, implementations may perform these updates on larger intervals (see xx for details).

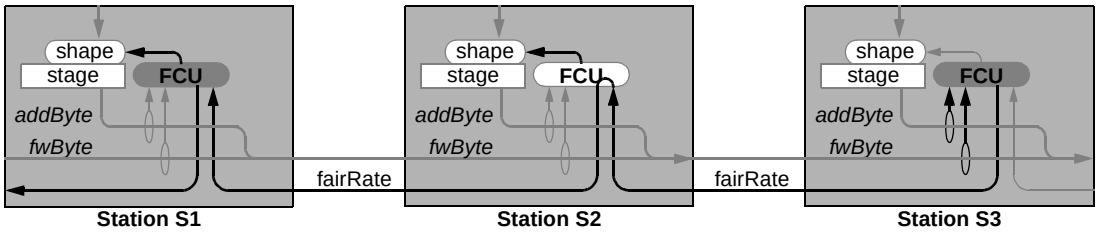


Figure 9.4—Fairness rate monitoring

NOTE—The rate monitors make no distinction between before congestion point and after congestion point traffic.

The fairness control unit (FCU) is responsible for monitoring these flows; the monitoring results are then reported by updating the reverse-flowing *fairRate* indication. In this example, *fairRate* fairness proceeds as follows:

a) Station S3 detects congestion and asserts a congestion-indicating *fairRate* value to station S2.
b) Station S2 is less congested and performs the following steps.
   1) Station S2 adjusts its shaper parameters to limit its fairness-eligible add traffic.
   2) Station S2 forwards the station S3 *fairRate* indication to station S1.
c) Station S1 is more congested and performs the following steps.
   1) Station S1 adjusts its shaper parameters to limit its fairness-eligible add traffic.
   2) Station S1 generates the *fairRate* indication for station S0 (not illustrated).

   Note that station S1 add rate is limited by the *fairRate* indication from station S2, not its own.

9.1.5 Avoidance controls

FIFO spillover avoidance controls (commonly referred to as avoidance controls) are generated by monitoring the STQ depth, as illustrated in Figure 9.5. While the specification is based on precise FIFO depth, coarse granularity of FIFO-depth measurements is allowed.

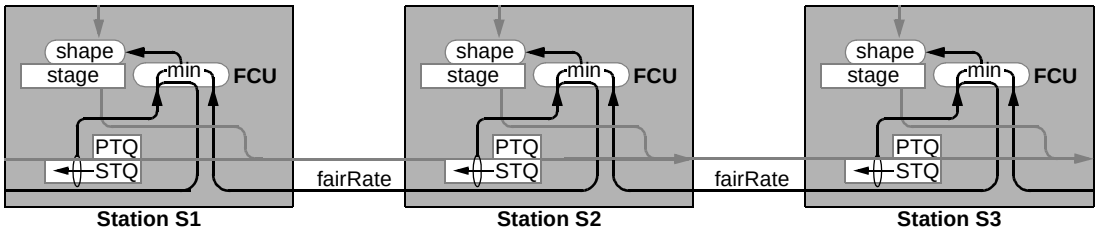


Figure 9.5—Spillover controls

Each station rate limits its classB (as well as classC) traffic when STQ thresholds are exceeded in the station FIFO or the station's downstream FIFO, as follows:

a) Station S2 stops its classB (as well as classC) add traffic after exceeding its avoidance threshold.
b) Station S2 inhibits STQ transmissions when downstream station S3's avoidance level is larger.

9.1.6 Fairness costing

The computation of *fairRate* value is based on *baseLine* and *fairLine* values, computed as illustrated Figure 9.6. When congested, the baseLine value approaches an averaged rate value; when uncongested, baseLine approaches an estimated effective value based on conflicting (transiting fairness eligible) traffic measurements.

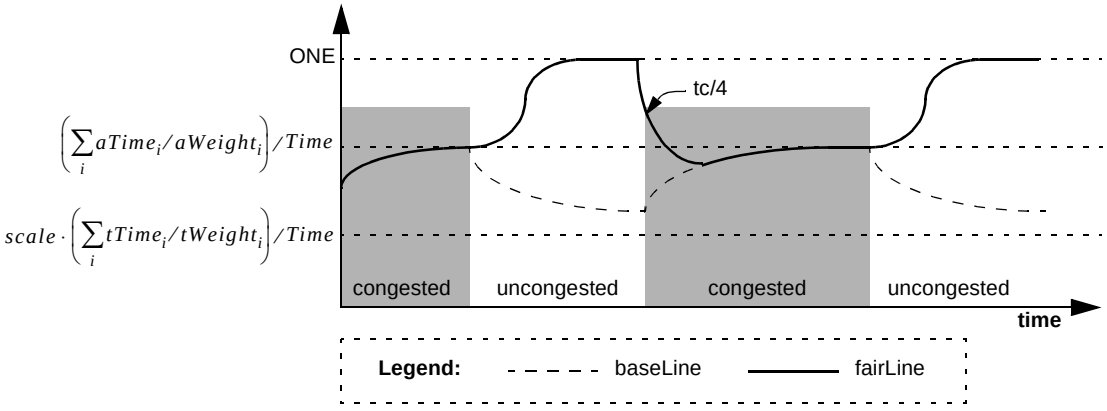


Figure 9.6—Asymtotes for *fairRate* computations

When congested, *fairLine* also approaches the averaged rate value; when uncongested, *fairLine* approaches the full-rate ONE value. The onset-of-congestion time constant is reduced to facilitate quick transitions between appropriate uncongested and congested phase values.

9.1.7 ClassC shaper behavior

The classC shaper is a small (typically 4-entry) array designed to manage multiple congestion domains, as illustrated in Figure 9.7. The multiple spatial-dependent shapers are intended to shape the traffic for array element W, X, Y, and Z, corresponding to increasingly remote ranges of destination stations. Traffic flowing to more remote locations also flows through intermediate shapers, so that simultaneous approval of shaperW, shaperX, and shaperY is necessary before transmissions within the range of shaperY.

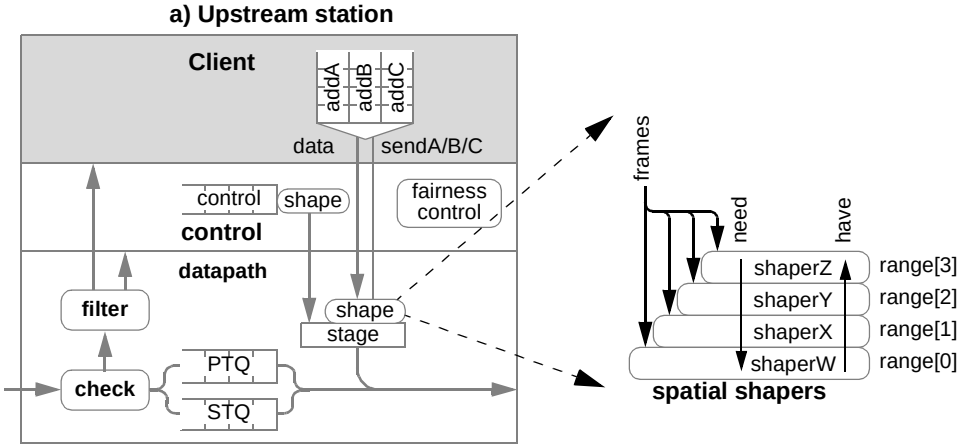


Figure 9.7—Spatial shaper components

The *high* indicator signals when longer-range shapers are enabled, with the intent of inhibiting debt-causing short-range transmissions while awaiting positive credit reports from shorter-range shapers. The *have*

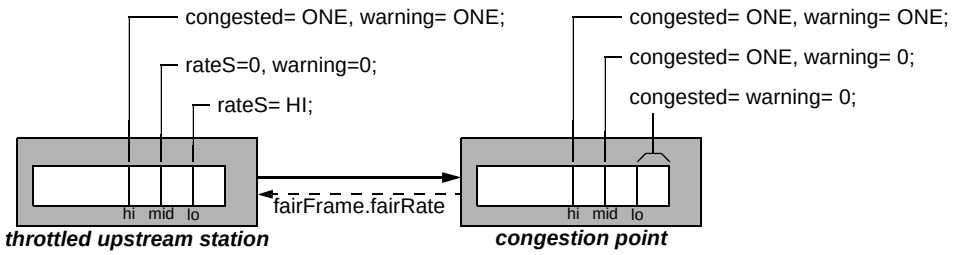
indicator signals when shorter-range shapers have positive credits, so a longer-range *passC* indication can be safely provided when simultaneous approvals become available.

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

An upstream shaper is used minimizes the response time while maintaining correct long-term behavior.
Limited spatial shapers are already defined in D2.2 Clause 9 (see addRate and addRateCongested).
The spatial-shaper concept is generalized to address separate fairness domains/subdomains.
This yields a more effective (than D2.2) solution, with similar hardware design costs.

9.1.8 Congestion levels

The fairness protocols are based on the reverse propagation of *fairRate* indications: a right-to-left data flow is associated with a left-to-right *fairRate* indication flow, as illustrated in Figure 9.8. The *fairRate* indication is nominally generated by a congestion point and throttles upstream stations(s), as illustrated in the right and left sides of Figure 9.8 respectively.



rateC= fairRate*weight;
okB= creditB>loLimit && creditS>loLimit;
okC= creditC>loLimit && creditS>loLimit;
okT= (warning<fairFrame.warning)

Figure 9.8—Congestion thresholds

The rate of the upstream station classC traffic is limited to *fairRate*weight*, thereby limiting transmissions to no more than the congested-station advertised restrictions. Both classB and classC traffic are also restricted based on the transit-queue depth, noted in Figure 9.8 with depth-dependent details illustrated in Figure 9.9.

An additional requirement is the use of a congestion-point warning indication. The warning inhibits upstream classB transmissions, to eliminate the possibility of having extended upstream classB/classC transmissions block critical downstream classA1 transmissions.

For simplicity and clarity, only dual-queue stations have been illustrated; variations of threshold conditions are applicable to single-queue station designs. Scaling factors (not illustrated) are also necessary to cope with non-equal station fairness weights.

9.1.8.1 Source-shaper rates

The *rateS* (rate of secondary) secondary-rate limiting parameter depends on the secondary transit queue (STQ) depth, as shown in Figure 9.9. Any monotonically increasing functions within the shaded polygon are allowed, although one of the two dotted-line functions should be implemented. One strategy is to limit the cumulative send-B and send-C transmissions, to bound the worst-case STQ delay. Another strategy is to back off the send-B and send-C transmissions when the secondary transit queue reaches a LO_THRESHOLD condition, so that sufficient STQ space is left for conflicting class-A transmissions.

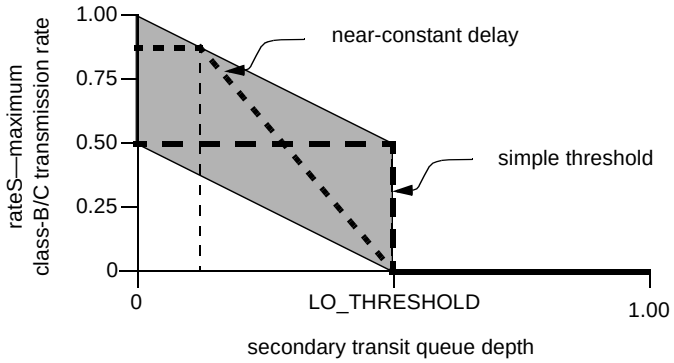


Figure 9.9—Depth-dependent source-shaper *rateS* parameter

NOTE—The intent is to give add-traffic precedence over STQ transmissions, while avoiding STQ transmission starvation, until the STQ filling could compromise the station's ability to ensure its classA1 transmission guarantees.

9.2 Terminology

9.2.1 Fairness MIB attributes

The fairness MIB attributes are listed in Table 9.1.

Table 9.1—Fairness MIB attributes

MIB attribute name(s)	Row	Cross reference	RPR name(s)
	1	TBD	fairnessWeight
	2	TBD	fairnessCoefficient
	3	TBD	stqThresholdLo
	4	TBD	stqThresholdHi
	5	TBD	stqThresholdMid

9.3 Fairness update state machine

9.3.1 Inputs

txFrame
The leading portion of the currently transmitted frame.

9.3.2 Local variables

addLevel
Measures the amount of client-supplied fairness-eligible traffic within each aging interval.
baseLine
A low-pass filtered approximation of the appropriate congested-state *fairLine* value.
(When non-congested, *fairLine* migrates towards a transit-flow based estimate).
fairLine
A low-pass filtered approximation of the *fairRate* value, normalized for internal accuracy.
(When non-congested, *fairLine* migrates towards FULL_RATE).
fairRate
A normalized representation of *fairLine*, for selective insertions in fairness-frame transmissions.
guess
An estimate of the number of contributing upstream stations.
topSide
Measures the amount of transit-supplied fairness-eligible traffic within each aging interval.
useLevel
Measures the amount of transit-supplied fairness-eligible traffic within each aging interval.

9.3.3 Shared variables

byteCount
A counter that is incremented by an external means, once after every data byte transmission.

9.3.4 Functions

AddByte()
Returns 1 when a transmitted data byte came from the client.
TransitByte()
Returns 1 when a transmitted data byte came from a transit buffer.

9.3.5 Rate measuring state table

Per-byte activities measure the number of bytes in each 100µs interval, as specified in Table 9.2.

Table 9.2—Rate measuring state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	AddByte() && FAIR(txFrame)	1	addCount += ONE/BYTES_PER_TICK;	MORE
	TransitByte() && FAIR(txFrame)	2	guess= MAX(1, SquareRoot(stations)/2);	
		3	useCount += (ONE/BYTES_PER_TICK) / (WEIGHT(frame)*guess);	
	—	4	—	
MORE	byteCount == 0	5	baseLine += FMUL3(addCount-baseLine, lpCoef, congestion)+ FMUL3(useCount-baseLine, lpCoef, ONE-congestion);	START
		6	topSide= MIN(ONE, 2*fairData); fairLine += FMUL3(baseData-fairLine, 4*lpCoef, congestion) : FMUL3(topSide-fairLine, lpCoef, ONE-congestion);	
		7	fairRate= fairLine>>15; addCount = useCount= 0;	
	—	8	—	
WAIT	byteCount==lastCount	9	—	WAIT
	byteCount== BYTES_PER_TICK	10	lastCount= byteCount= 0;	START
	—	11	lastCount= byteCount;	

```
#define ONE 0X80000000
#define FMUL3(x,y,z) ((((uInt8)x*(uInt8)y>>30)*(uInt8)z)>>30)
#define WEIGHT(frame) (DataBaseEntry[frame.ttlBase-frame.ttl].weight)
#define FAIR(frame) ((frame.sc==CLASS_B&&frame.fe==1)||frame.sc==CLASS_C)
```

Row 9.3-1: Count the number of bytes .
Row 9.3-2: The *guess* value represents a logrithmic mean of min/max transmit-station possibilities.
Row 9.3-3: Normalize useCount by a precomputable inverse of the source-station weight and *guess* values.
Row 9.3-4: Other transmitted bytes (or byte-long time delays) are not counted.

At the end of every 10us, the following computations are performed:
Row 9.3-5: The precise *baseData* value approaches *addCount* when congested; *useCount* otherwise.
Row 9.3-6: The precise *fairData* value approaches *baseData* when congested; *topSide* otherwise.
Limiting *topSide* to no-more-than twice the *fairData* value defers large increases of small *fairData* values.
Row 9.3-7: The fairRate value is computed and byte-count values are cleared.
Row 9.3-8: No filtering operations are performed for most data-byte transmissions.

Row 9.3-9: Wait for a *byteCount* value change, indicating the next byte has been transmitted.
Row 9.3-10: Wrap the *byteCount* value when the aging interval has been reached.
Row 9.3-11: Increment the *byteCount* value when the aging interval has not been reached.

9.4 Fair shaper state machine

9.4.1 Inputs

clientC
: Set to 1 when a fairness-eligible frame has been sent by the client.

hiLimitC
: An upper bound for the level of accumulated class-C credits.

time
: A constantly updated unsigned value that supplies the current time value.

txFrame
: The leading portion of the currently transmitted frame.

9.4.2 Local variables

next
: The expected next level for *myCreditC*, before application of *hiLimitC* saturation limits.

lastTime
: The last time when shaper credits were updated.

9.4.3 Shared variables

myCreditC
: The level of accumulated transmission credits.

9.4.4 Functions

SizeOf(frame)
: Returns the size of its frame-data-structure argument, in bytes.

9.4.5 Fair shaper state table

The classC shaper limits the client-supplied classC transmissions to their allocated limits, using a small array (typically 4) of state machines specified in Table 9.3.

Table 9.3—Fair shaper state table

Conditions		Row	Results	
state	**input condition**		**actions**	**state**
START	clientC && HOPS(txFrame)>=myPlace	1	myCreditC –= SizeOf(txFrame);	START
	(time–lastTime) >= TICK	2	next= myCreditC + rateC*TICK;	
		3	myCreditC= MIN(hiLimitC, next);	
		4	lastTime= time;	
	—	5	—	

#define HOPS(frame) (255–frame.ttl)
#define hiLimitC 3*MTU
#define rateC (linkBw*MIN(maxRateC, myWeight*(myLevel/65535.)))

Row 9.3-1: The shaper credits are reduced whenever within-range frames are transmitted.
Row 9.3-2: Credits accumulate at the end of every time-tick interval.
Row 9.3-3: Credits accumulation is clamped at a *hiLimit* upper bound.
Row 9.3-4: Shaper updates are scheduled for the next time-tick interval.
Row 9.3-5: No shaper updates are available to be performed.

TBDs—
Clear credits after 2*MTU of client's lower-precedence/nonexistent traffic since send was enabled.
For classA and classB shapers, *place* is first location within the range, shaper[0] is nonexistent.

9.4.6 Fair shaper coupling

The classC shaper credits are coordinated, as specified in Table 9.4. When executed on *range[n]*, the values of *infoLo*, *info*, and *infoHi* represent the shaper parameter groups of *range[n–1]*, *range[n]*, and *range[n+1]* respectively.

Table 9.4—ClassC shaper credit coordination

Conditions		Row	Results	
state	**input condition**		**actions**	**state**
START	n=0	1	—	FIRST
	n=N-1	2	—	FINAL
	—	3	—	CORE
FIRST	info.credits>=2*loLimit	4	info.have= MOST, info.ends= infoHi.place–1;	START
	infoHi.need==0 && info.credits>=loLimit	5	info.have= MORE, info.ends= infoHi.place–1;	
	infoHi.need==1 && info.credits>=loLimit	6	info.have= MORE, info.ends= 0;	
	—	7	info.have= LESS, info.ends= 0;	
FINAL	infoLo.have>LESS && info.credits>=loLimit	8	passC= 255;	
	info.credits >= loLimit	9	info.need= 1, passC= infoLo.ends;	
	—	10	passC= infoLo.ends;	
CORE	infoLo.have==MOST && info.credits>=2*loLimit	11	info.have= MOST, info.ends= infoHi.place–1	
	infoHi.need==0 && infoLo.have==MORE && info.credits>=loLimit	12	info.have= MORE, info.ends= infoHi.place–1	
	infoHi.need==1 && infoLo.have==MORE && info.credits>=loLimit	13	info.have= MORE, info.ends= infoLo.ends;	
	—	14	info.have= LESS	

#define loLimit MTU
enum {LESS, BASE, PLUS, BEST};

Row 9.3-1: Row 9.3-2: Row 9.3-3: Because of shapers receive *have* and *need* from preceding and following shapers, the first, intermediate, and final shapers have slightly different sendC contribution behaviors.

Row 9.3-4: Enable first-shaper transmissions when credits exceed the 2**loLimit* threshold.
Row 9.3-5: Conditionally enable first-shaper transmissions when credits exceed the *loLimit* threshold.
Row 9.3-6: Pass first-shaper transmit-permission when credits exceed the *loLimit* threshold.
Row 9.3-7: Otherwise, first shaper transmit-permission is denied.

Row 9.3-8: Enable final-shaper transmissions when preceding credits exceed the *loLimit* threshold.
Row 9.3-9: Disable final-shaper transmissions until preceding credits exceed the *loLimit* threshold.
Row 9.3-10: Otherwise, final shaper credits accumulate toward threshold values.

Row 9.3-11: Enable core&following transmissions when credits reach the 2**loLimit* threshold.
Row 9.3-12: Enable core transmissions when local&lowside credits reach the *loLimit* threshold.
Row 9.3-13: Enable following transmissions when local&lowside credits reach the *loLimit* threshold.
Row 9.3-14: Otherwise, disable following transmissions.

TBD—
First range entry fixed with *place*=0 & *maxRate*, others use *place*=n & *fairRate* as set by Table 9.7.

9.5 Dense profiling state machine

9.5.1 Dense profile arrays

Multichoke fairness can be supported by a large array of rating values, where each level value corresponds to a distinct hop-count distance. This approach is conceptually simple, but requires a 255-entry memory array to support each of the 255 possibly configured stations.

The receipt of a fairness frame changes rating levels associated with its hop-count distance. Other changes are sometimes performed, to maintain the monotonic nature of the cost-factor levels. Thus, a level reduction for hop-count 2 can also reduce levels for hop-count distances {3,4,5}, as illustrated in the left side of Figure 9.10. Similarly, a level increase for hop-count 6 can increase levels for hop-count distances {2,3,4,5}, as illustrated in the right side of Figure 9.10.

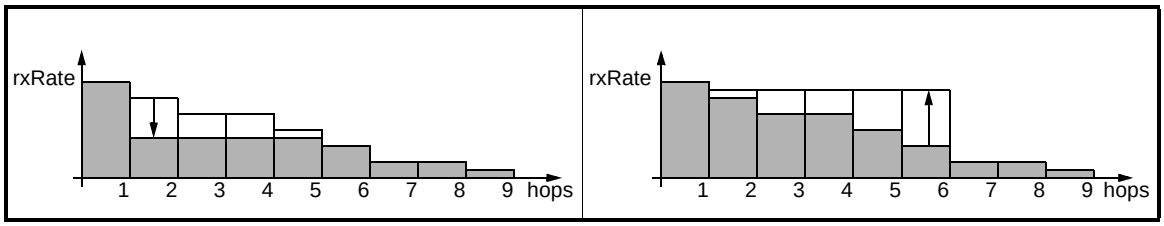


Figure 9.10—Dense *fairRate* profile arrays

9.5.2 Inputs

rxFrame
The received fairness frame.

9.5.3 Local variables

me.hops
The index for the fine-grained *fairRate* profile entry
me.rate
The assumed classC rate-limiting value, based on previously observed applicable *fairRate* values.

9.5.4 Dense profile states

The update algorithm can be phrased in terms of the conditions that determine which array elements are set to the reported *fairRate* value, as specified in Table 9.5.

Table 9.5—Dense profile states

Conditions		Row	Results	
state	**input condition**		**actions**	**state**
START	rxFrame.scope==me.hops	1	me.rate= rxFrame.fairRate;	START
	rxFrame.scope>me.hops && rxFrame.fairRate>me.rate	2		
	rxFrame.scope<me.hops && rxFrame.fairRate<me.rate	3		
	—	4	—	

Row 9.5-1: The scope corresponds to this entry; accept the source's reported *fairRate* value.
Row 9.5-2: The scope is beyond this entry; accept a source's higher *fairRate* value.
Row 9.5-3: The scope is before this entry; accept a source's lower *fairRate* value.
Row 9.5-4: Otherwise, no adjustments are performed.

9.6 Sparse profile state machine

9.6.1 Sparse profile overview

Multichoke fairness can also be supported by a small array of level values, where each element corresponds to a nonoverlapping range of hop-count distances. This approach is conceptually more complex, but requires less memory and fewer shapers to support each of the 255 possibly configured stations.

The receipt of a fairness frame changes rating levels associated with its hop-count range. Other changes are sometimes performed, to maintain the monotonic nature of the rating levels. Thus, a level reduction for range 2 can also reduce levels for ranges distances {3,4}, as illustrated in the left side of Figure 9.11. Similarly, a level increase for range 4 can increase levels for hop-count distances {2,3}, as illustrated in the right side of Figure 9.11.

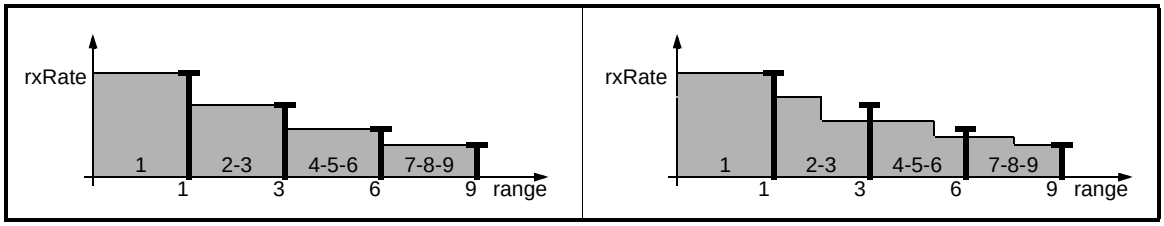


Figure 9.11—Sparse profile structure

9.6.2 Inputs

lo.hops
: The distance associated with the furthest preceding-range location.

lo.rate
: The level associated with the preceding *lo.hops* location.

rxFrame
: The received fairness frame.

9.6.3 Shared variables

me.hops
: The index for the fine-grained *fairRate* profile entry

me.level
: The *fairRate* level associated with the adaptive *me.place* congestion location.

me.place
: An adaptive location associated with a significant change in the monitored *fairRate* level.

me.rate
: The assumed classC rate-limiting value, based on previously observed applicable *fairRate* values.

9.6.4 Sparse profile states

For example, Table 9.6 illustrates how a sparse *fairRate* profile could be associated with four range-array elements, as specified in Table 9.6.

Table 9.6—Sparse profile states

Conditions		Row	Results	
state	**input condition**		**actions**	**state**
START	me.hops==rxFrame.scope	1	me.rate= rxFrame.fairRate;	NORM
	rxFrame.scope>me.hops && rxFrame.fairRate>me.rate	2		
	rxFrame.scope<me.hops && rxFrame.fairRate<me.rate	3		
	—	4	—	
NORM	me.level>lo.rate || me.level<me.rate	5	me.level= me.rate, me.place= me.hops;	NEXT
	—	6	—	
NEXT	rxFrame.scope<=lo.hops || rxFrame.scope>me.hops	7	—	START
	me.place==rxFrame.scope	8	me.level= rxFrame.fairRate, me.place= rxFrame.scope;	
	rxFrame.scope>me.place && rxFrame.fairRate>=me.level	9		
	rxFrame.scope<me.place && rxFrame.fairRate<me.level	10		
	rxFrame.scope<me.place && rxFrame.fairRate<THRESHOLD	11		
	—	12	—	

#define THREHOLD MIN((lo.rate+me.rate)/2, (me.rate*3)/2)

Row 9.6-1: The scope corresponds to this entry; accept the source's reported *fairRate* value.
Row 9.6-2: The scope is beyond this entry; accept a source's higher *fairRate* value.
Row 9.6-3: The scope is before this entry; accept a source's lower *fairRate* value.
Row 9.6-4: Otherwise, no adjustments are performed.

Row 9.6-5: Clamp intermediate levels to within the range edge-condition values.
Row 9.6-6: Continue checks for within-range level adjustments.

Row 9.6-7: Skip the reports from outside-the-range hop-count distances.
Row 9.6-8: If the report matches the within-range place location, update this range's level value.
Row 9.6-9: When scope is beyond placement and *fairRate* is above *level*, accept the *fairRate* value.
Row 9.6-10: When scope is before placement and *fairRate* is above *level*, accept the *fairRate* value.
Row 9.6-11: When scope is before placement and *fairRate* is below threshold, accept the *fairRate* value.
Row 9.6-12: Otherwise, no *fairRate* adjustments are performed.

9.7 Multichoke congestion reports

9.7.1 Opposing fairness frames

The fairness protocol is intended to efficiently support nonblocking virtual queues, by distributing congestion information with a per-hop granularity. Flow control information for station S1 transfers is returned from stations S6-through-S2, as illustrated in the left side of Figure 9.2.

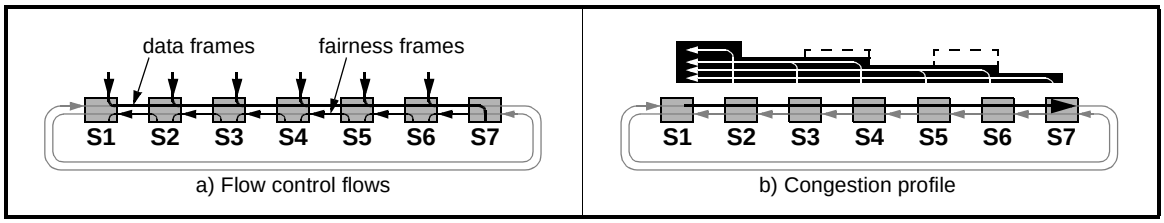


Figure 9.12—Flow controlled stations

This flow control information is intended to provide a flow-rate profile of the downstream stations, as illustrated by the black histogram in the right side of Figure 9.2. That profile is monotonic decreasing plot of weighted congestion flow-rates versus hop-count distance. Within this station S1 histogram, the lightly congested conditions (shown with dotted lines) are masked by more severe closer congestion conditions.

Station S1 uses the flow-rate profile to limits its flow to no more than the worst-case (i.e. the smallest) flow-rate between itself and its selected destination station, on a per frame basis. The profile allows transmissions to be evaluated on a hop-count distanced basis. Thus, moderate-rate transmissions can be allowed over the S1-to-S4 or S1-to-S6 paths, despite flow-rate blocking of lower-rate S1-to-S7 transmissions.

9.7.2 Flow-rate indications

Such flow-rate profiles could be generated by having each station broadcast its congestion information in the upstream station. Such distribution strategies would unnecessarily suffer from excessive overheads, since the overhead would increase in proportion to the number of stations *N*. Sending fairness messages less frequently, could compensate for the cumulative effect of broadcast messages, but would have the undesirable effect of increasing the message-delivery latencies and there the system response times.

A different approach is therefore taken: flow-rate messages are point-to-point periodic messages. These periodic messages pass into a station, the flow-rate information from that station is merged, and that merged information is sent to the next downstream station. The merging process is not lossless (the larger S3-to-S4 flow-rate remains hidden), but maintain the important monotonic-decreasing flow-rate profile information.

The key features for supporting this feature include the following:

a) Dithered start. As currently defined, fairness messages “start” at a relieved congestion point. Dithering the starting point, to start at a pseudo-random point, this would light-up dark spots.
b) Supplemented. Each fairness message communicates its “start” location within fairness frames.

9.7.3 Dithered restart sequences

To ensure a well distributed set of fairness-frame starting locations, a selected station specifies a dithered set of fairness-sampling restart locations for well-aged fairness frames. After aging, the restart selection process is performed in a distributed fashion, as illustrated in Figure 9.13 and desribed below.

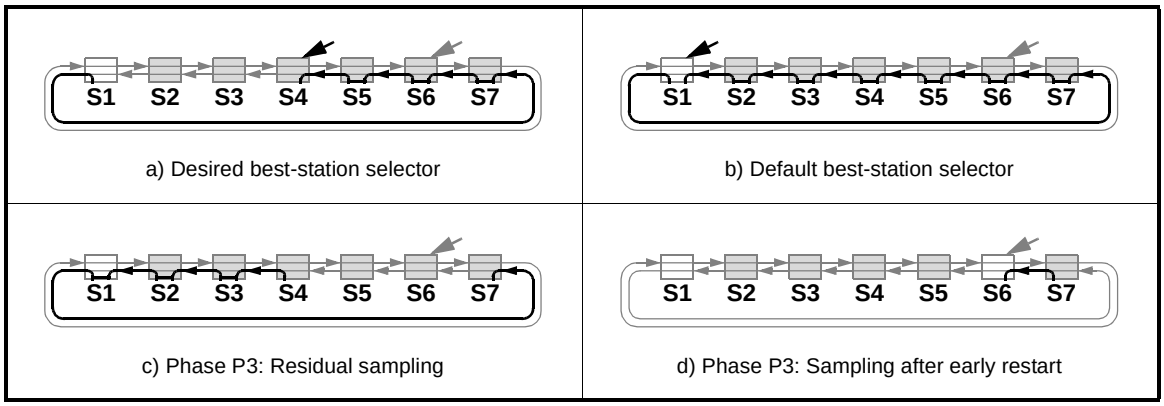


Figure 9.13—Distributed flow-control restarts

a) The restart location is nominally specified by a best-selection selector.
b) On a closed ring, in the absence of a best-selector, the restart location is specified at the source. On an open ring, the restart location is specified by the edge stations (not illustrated).
c) If the fairness frame is updated before the restart location, the *ttl*-relative restart location is updated.
d) The fairness frame is removed at its dithered best-selector specified restart point.

9.8 Fairness update state machine

9.8.1 Inputs

rxFrame
The received fairness frame

9.8.2 Outputs

txFrame
The transmitted fairness frame

9.8.3 Local variables

myRingID
The ringletID of this port.
myEdgeStatus
Set to EDGE when this topology has labeled this as an edge station.
myFairRate
The *fairRate* value associated with fairness-eligible transmissions from port.
myMacAddress
The MAC address associated with this station.
myMacIsLargest
The MAC address associated with this station is the largest one found in the topology database.
myStationCount
The count of attached stations (not the ports, which could be twice as large when wrapped).

9.8.4 Functions

FetchRequest()
Returns the most recent fairness frame, or NULL if that fairness frame has already been returned.

9.8.5 Fairness aging state table

The detailed definition of these fairness frame updates is specified in Table 9.7.

Table 9.7—Fairness aging state machine

Conditions		Row	Results	
state	**input condition**		**actions**	**state**
FIRST	rxFrame.age==OLD && rxFrame.more==0 && rxFrame.less==0	1	—	LAST
	rxFrame.ri==myRingID && myFairRate<=rxFrame.fairRate	2	—	THIS
	rxFrame.ri==myRingID && rxFrame.saCompact==myMacAddress	3	—	SELF
	rxFrame.ri!=myRingID && myFairRate<=rxFrame.fairRate	4	—	THAT
	—	5	—	START
THIS	rxFrame.more==1&& SCOPE(rxFrame)>=myStationCount	6	txFrame.scope= 0, txFrame.age= OLD, SET(txFrame);	TEST
	SCOPE(rxFrame)<myStationCount	7	txFrame.scope += HOPS(rxFrame), SET(txFrame);	
	—	8	txFrame.scope= 0, txFrame.more=1, SET(txFrame);	
SELF	rxFrame.less==0	9	txFrame.scope= txFrame.more= 0, txFrame.less= 1, SET(txFrame);	TEST
	rxFrame.less==1	10	txFrame.scope= txFrame.more= 0, txFrame.age= OLD, SET(txFrame);	
THAT	rxFrame.age==OLD	11	—	TEST
	rxFrame.age!=0 && rxFrame.less==0	12	txFrame.scope= 0, txFrame.more= 1, txFrame.age= rxFrame.age + 1, SET(txFrame);	
	—	13	txFrame.scope = txFrame.more= 0, txFrame.age= rxFrame.age + 1, SET(txFrame);	
TEST	rxFrame.age!=OLD	14	—	START
	myEdgeStatus==EDGE	15	—	POST
	myMacIsLargest	16		
	rxFrame.saCompact==myMacAddress	17		
	SCOPE(rxFrame)>=myStationCount	18		
	—	19	txFrame.scope += HOPS(rxFrame),	START

Table 9.7—Fairness aging state machine *(continued)*

Conditions		Row	Results	
state	**input condition**		**actions**	**state**
TEST	count >= COUNT	20	count= 0;	AGED
	count >= (COUNT-1) && myEdgeStatus==EDGE && myRingID==RINGLET_0;	21		
	—	22	txFrame.age= txFrame.less= 0, count+= 1;	START
AGED	HASH(code) != 0	23	txFrame.scope= HASH(count), txFrame.more= txFrame.less= 0; txFrame.age= OLD, SET(txFrame);	START
	—	24	—	CLEAR
LAST	HOPS(rxFrame) >= rxFrame.scope	25	—	CLEAR
	myFairRate <= rxFrame.fairRate	26	txFrame.scope= rxFrame.scope-HOPS(rxFrame), SET(txFrame);	START
	rxFrame.ri==myRingID && rxFrame.saCompact==myMacAddress	27		
	—	28	—	
START	(rxFrame=FetchRequest())==NULL	29		START
	rxFrame.ttl<=1	30	—	CLEAR
	—	31	txFrame= rxFrame;	FIRST
CLEAR	—	32	txFrame.scope= txFrame.age= 0, txFrame.more= txFrame.less= 0, SET(frame);	START

```
#define OLD 3
#define HOPS(frame) (255–frame.ttl)
#define SCOPE(frame) (HOPS(frame)+(uInt2)(frame.scope))
#define SET(frame) (frame.ttl= 255, frame.ri= myRingID, frame.sa= myMacAddress, frame.fairRate= myFairRate)
```

Row 9.7-1: These fairness frames are marked for random destruction; process them distinctively.
Row 9.7-2: The *fairRate* value decreases when passing through a more congested same-ringlet station.
Row 9.7-3: The *fairRate* value increases when passing through a less congested source station.
Row 9.7-4: The *fairRate* value decreases when passing through more congested opposing ringlet station.
Row 9.7-5: Otherwise, the fairRate value is not affected.

Row 9.7-6: Twice-aged fairness frames are marked for old-age retirement.
Row 9.7-7: Youthful fairness frames countdown the time before retirement.
Row 9.7-8: Aging fairness frames survive middle-age, progressing towards retirement.

Row 9.7-9: Source stations mark the first time that *fairRate* is returned and increased.
Row 9.7-10: Twice marked frames are marked for old-age retirement.

Row 9.7-11: Aged alternate-run fairness frames have already been marked for retirement.
Row 9.7-12: Aged never-increased alternate-run fairness frames are marked scope-valid.
Row 9.7-13: Other alternate-run fairness frames are marked scope-invalid.

Row 9.7-14: Pre-retirement frames continue advancing towards retirement.
Row 9.7-15: Each edge is responsible for resuming and dithered recycling retired fairness frames.
Row 9.7-16: In the absence of edges, the *best* port is responsible for resuming and dithered recycling.
(The best port only exists on a closed-loop topology and is self-selected based on the largest MAC address.)
Row 9.7-17: The source port accepts recycling responsiblities, when *fairRate* decreases.
Row 9.7-18: All ports accepts recycling timeout responsiblities, when *fairRate* never decreases.
Row 9.7-19: The scope value supports a recycling timeout, when *fairRate* never decreases.

Row 9.7-20: Most stations recycle one of every COUNT frames for dithered recycling.
Row 9.7-21: The ringlet1 edge station recycles one of every COUNT-1 frames for dithered recycling.
Row 9.7-22: Frames that are not marked for destruction are recycled with inherited values.

Row 9.7-23: A frame's dithered recycling point is marked by its *scope* field value.
Row 9.7-24: If this scope field value is zero, the frame is recycled immediately.

Row 9.7-25: Frames are recycled when their recycling point is reached.
Row 9.7-26: At early congestion-increase points, recycling frames inherit new *ttl*-relative recycling counts.
Row 9.7-27: At early congestion-decrease points, recycling frames inherit new *ttl*-relative recycling counts.
Row 9.7-28: Otherwise, unaffected fairness frames continue towards their recycling point.

Row 9.7-29: Wait for fairness frames to become available.
Row 9.7-30: Suspect fairness frame are discarded when the *ttl* (time to live) decrements to zero.
Row 9.7-31: Valid fairness frames are processed.

Row 9.7-32: Initialized fairness frames are cleared and *fairRate* is set to that of this source station.

9.9 Single-queue congestion indications

The *fairRate* indication is intended to inhibit upstream classC traffic when downstream stations are unfairly blocked. Other related feedback mechanism are necessary to ensure progress of downstream classB traffic that can be blocked by upstream classC traffic and classA traffic that can be blocked by upstream classB/classC traffic. This feedback is intended to control upstream transmissions based on downstream subclassA1/classB congestion levels, without disturbing unrelated classC fairness feedback controls.

The details of these feedback depend on the single-queue or dual-queue nature of the MAC.

Editors' Notes: *To be removed prior to final publication.*
The following text is highly preliminary and could use refinements.

For a single-queue MAC, congestion is indicated when reserved plus passthrough traffic exceeds 90% of the link capacity.

9.10 Fairness frames

9.10.1 Fairness frame format

The format of the proposed fairness frame is shown in Figure 9.14.

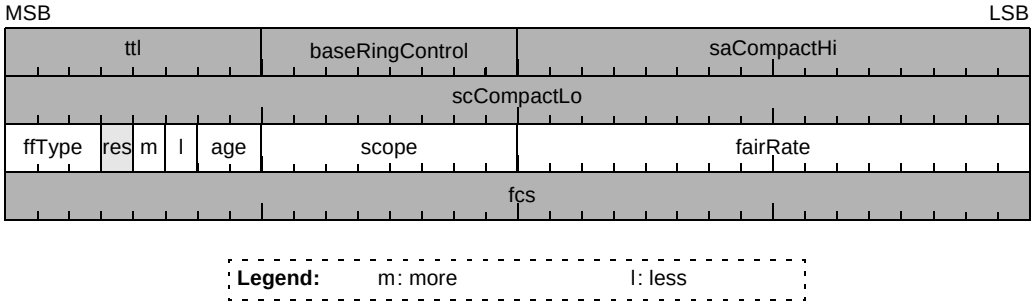


Figure 9.14—Fairness frame format

9.10.1.1 *ffType*: The 3-bit (fairness frame type) field that distinguishs this from other fairness frame formats.

9.10.1.2 *more*: A 1 value indicates the scope effectively includes all stations; 0 indicates otherwise.

9.10.1.3 *less*: A 1 value indicates the *fairRate* has been reduced by its source; 0 indicates otherwise.

9.10.1.4 *age*: A 2-bit field that is incremented each time *fairRate* is increased on the opposing ringlet. A full-scale value also indicates the fairness has served its purpose and can be selected for dithered restart.

NOTE—The combination of {*more*==0, *less*==0, *age*=3) cannot be created through the fairness frame aging sequence, and is therefore used to identify overaged frames destined for dithered restart, at a scope-specified distance.

9.10.1.5 *scope*: The unsigned 8-bit (congestion scope) field set by the FCU and passed to the transmitter. The startup FCU sets *scope* to 0; other FCUs effectively increment *scope*. See xx for details.

9.10.1.6 *fairRate*: A 16-bit field that communicates bandwidth constraints of congested stations. See xx for details.