SSA: simple scheduling algorithm for resilient packet ring networks

F. Alharbi and N. Ansari

Abstract: The resilient packet ring (RPR), defined under IEEE 802.17, has been proposed as a high-speed backbone technology for metropolitan area networks. RPR is introduced to mitigate the underutilisation and unfairness problems associated with the current technologies SONET and Ethernet, respectively. The key performance objectives of RPR are to achieve high bandwidth utilisation, optimum spatial reuse on the dual rings, and fairness. The RPR standard implements three traffic classes: Class A, Class B, and Class C. The RPR MAC has one queue for each traffic class. A potential performance limitation is associated with the head-of-line blocking. When the MAC uses a single FIFO to buffer frames awaiting access, a packet that is traversing through a congestion point may block transmission of other packets destined to a point before the congestion. The use of virtual destination queues (VDQs) to avoid the head-of-line blocking is introduced. Different bandwidth allocation policies are discussed to assign rates to VDQs. Finally, a bandwidth allocation policy is proposed, which would achieve the maximum utilisation at a very low complexity.

1 Introduction

Rings are the most prevalent metro technologies because of their protection and fault tolerance properties, but the current metropolitan ring networking technologies exhibit several limitations. In a SONET ring, each node is granted with the minimum fair share, but it is not possible to reclaim the unused bandwidth; moreover, 50% of the potentially available bandwidth is reserved for protection, thus resulting in poor utilisation. Alternatively, Gigabit Ethernet assures full statistical multiplexing at the expense of fairness.

Resilient packet ring (RPR) [1] shares SONET's ability in providing fast recovery from link and node failures in addition to inheriting the cost and simplicity of Ethernet. Like SONET/SDH, RPR [2] is a ring-based architecture consisting of two optical rotating rings: one is referred to as the inner ringlet, and the other the outer ringlet (Fig. 1). Both rings can be used for transporting data packets. In RPR [3, 4], packets are removed from the ring at the destination so that different segments of the ring can be used at the same time for different flows; as a result, the spatial reuse feature is achieved. RPR defines three service classes for user traffics: Class A which has guaranteed rate and jitter, Class B with a committed information rate (CIR) and bounded delay and jitter, and the best effort traffic (Class C). The current RPR standard uses a single FIFO for each class at the ingress point (Fig. 2), and thus the head-of-line blocking is a potential problem. Optionally, the MAC may implement virtual destination queues (VDQs) to avoid the head-of-line blocking. In this paper, we discuss the limitation of the per-class queue scheme, and introduce the VDQ scheme, which serves these VDQs by using a unique and scalable bandwidth allocation algorithm.

IEE Proceedings online no. 20045232

doi:10.1049/ip-com:20045232

Paper first received 4th October 2004 and in final revised form 23rd November 2005

The authors are with the Advanced Networking Lab., Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA

E-mail: nirwan.ansari@njit.edu

2 The resilient packet ring

2.1 Flow control in RPR

The flow control in RPR is achieved by enabling a backlogged node (Fig. 3*a*) to send the fairness message according to its local measurements to the upstream nodes to throttle ingress data rates in order to eliminate the state of congestion and apply fairness among all the participating nodes [5–7].

RPR adopts the ring ingress aggregated with spatial reuse (RIAS) fairness concept [5], where the level of traffic granularity at a link is defined as an ingress-aggregated (IA) flow, i.e. the aggregate of all flows originated from the same node but destined to different nodes. At the state of congestion, all nodes should be able to send the same amount of data on the congested link relative to the other nodes.

2.2 Scheduling

To ensure hardware simplicity and that the transit path is lossless, the RPR node [5] does not include per–ingress or per-flow queues on the transit path; instead, it supports two

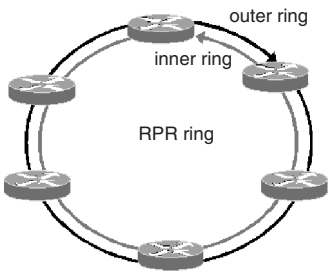

Fig. 1 *The resilient packet ring*

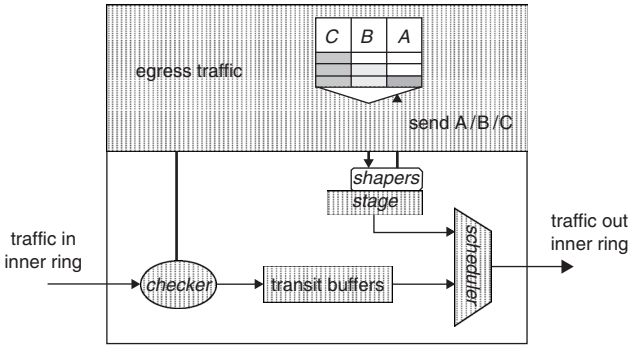

Fig. 2 *Station traffic shaping*

scheduling modes. In the single–queue mode (Fig. 3*b*), the transit path is a single FIFO and the transit traffic has a strict priority over the station traffic. However, in the dual-queue mode (Fig. 3*c*), the transit path consists of two queues: primary transit queue (PTQ) for Class A traffic, and secondary transit queue (STQ) for Class B and Class C traffic; in this mode, PTQ will be served first. When PTQ is empty, STQ has strict priority over the station traffic when the queue length exceeds the STQ threshold; otherwise, the station traffic is served in the following order: Class A, then Class B. If the station (node) has no Class A or B traffic, then Class C traffic will be served.

2.3 Traffic shaping

Class A traffic has a guaranteed rate, and the unused Class A bandwidth cannot be reclaimed. Class B traffic is a committed information rate (CIR). Thus, we omit the discussion of Class A and Class B traffic. Throughout the rest of the paper, we consider Class C in which each node uses the unreserved bandwidth and reclaims Class B unused bandwidth.

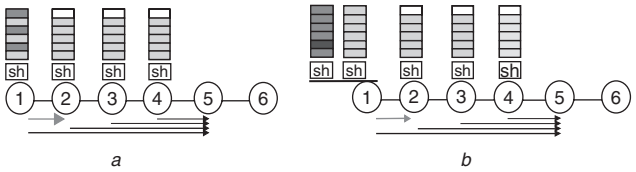

Fig. 4 *Simple scheduling scenario*
a Without VDQ
b With VDQ

The RPR node uses one queue per class for the station traffic. To show the limitation of this architecture, we consider the simple scheduling scenario (Fig. 4*a*) where all flows are Class C traffic. When virtual destination queues (VDQs) are not used, flow (1,2) is unnecessarily throttled and delayed owing to the congestion that flow (1,5) is experienced at link 4. Alternatively, with VDQs (Fig. 4*b*), flow (1,2) will be able to reclaim the unused bandwidth at link 1.

The benefit of using the VDQ scheme is obvious, but the challenge is how to manage these queues to maximise the utilisation and maintain fairness at the ring level. This issue will be discussed in the following Sections.

3 The VDQ scheme

To avoid the head-of-line blocking problem associated with the per-class queue scheme, we introduce the scheme

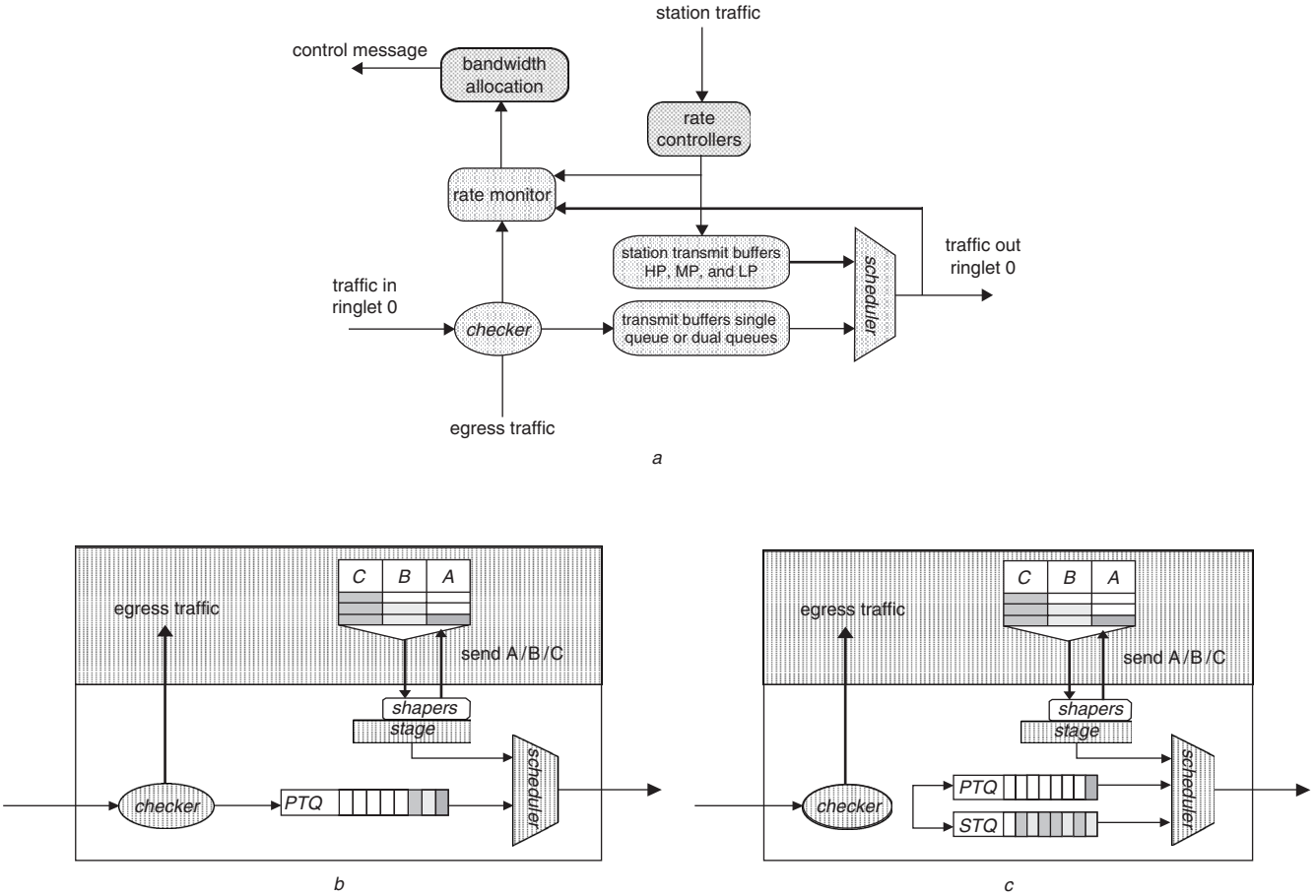

Fig. 3 *The node architecture*
a The generic architecture
b Single queue
c Dual queue

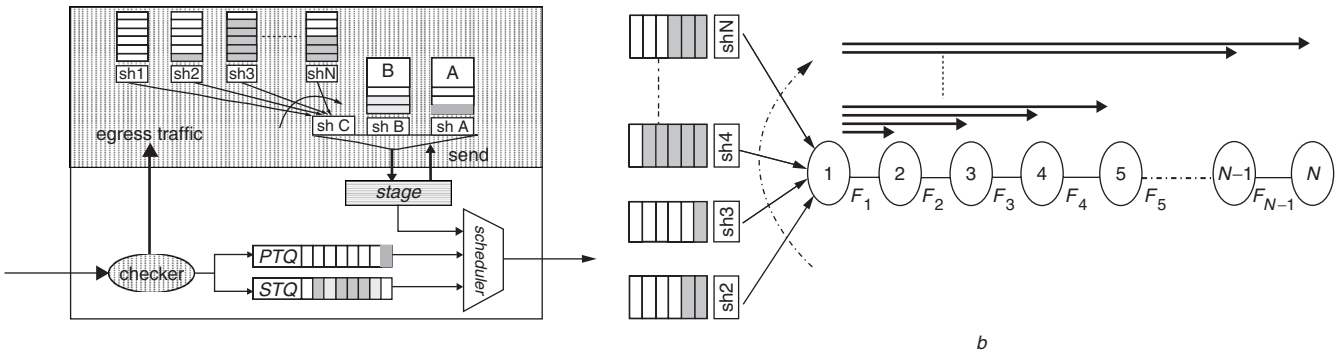

Fig. 5 *The VDQ scheme*
a The proposed RPR node architecture
b A general scheduling scenario

illustrated in Fig. 5*a*. Here we only consider Class C, where each node uses per destination queue.

Now, we assume that each node is aware of the per-source fair rates F_i at all the links. To make sure that a station does not exceed its fair rate at each link, each VDQ is controlled separately by its traffic shaper.

The above scheduling scheme would require a per source-destination allocation at the ingress point. For better understanding, we consider the general case as illustrated in Fig. 5*b*.

Let r_k be the estimated demand from node 1 to node k over the period T s. Node 1 will use these demands along with the latest received fair rate from each link i, F_i, to adjust the rate of each VDQ shaper as follows.

Define f_i as the rate allocated for every flow from node 1 to node k ($k = i+1, i+2, \ldots, N$) that traverses node i.

The goal of the allocation policy is to maximise f_i for $i = 1, 2, 3, \ldots, N-1$ subject to the constraint

$$\sum_{\substack{k = i+1, i+2, \ldots, N \\ (i.e., \text{ traversing link } i)}} r_k \leq F_i \qquad (1)$$

The allocation policy has to make sure that the sum of all flows from the same ingress point, destined to different nodes, traversing link i, do not exceed the per-source fair rate at link i, F_i.

The MAC will set the VDQ shaper to the minimum f_i along the path from the source to the destination.

4 Bandwidth allocation policy

4.1 Round-robin allocation policy

In this Section, we consider three allocation policies. In the first policy [8], node 1 maps the received fair rate F_i into a counter, credit [i], which represents the number of bytes node 1 can transmit over link i during the next T s. The virtual destination queues (VDQs) are served in a round robin fashion. When the VDQ [j] has a packet to be sent to destination j, the procedure illustrated in Fig. 6 [8] will be executed to determine the bottleneck link. The procedure returns the link number in which node 1 has no more credit and not allowed to transmit through. Thus, all destinations beyond the limited link are unreachable. Alternatively, if the destination is before the limited link the packet will be transmitted and all links traversed by the packet will have their credit [i] decreased by the packet size.

It is clear that this policy is very complex and not scalable as the number of links in the ring increases. Moreover, it is not fair owing to the fact that the packet size is not fixed.

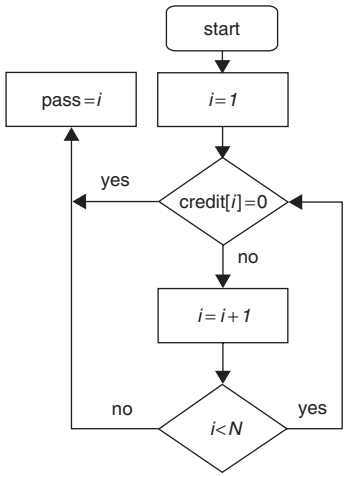

Fig. 6 *The credit based policy [8]*

4.2 Equal allocation policy

The second policy is the equal allocation policy. The link fair rate is divided by the number of flows traversing that link.

Define m_i as the number of flows originated from node 1 that traverses link i. Then, the per flow fair rate is

$$f_i = \frac{F_i}{m_i} \qquad (2)$$

The equal allocation (Fig. 7*a*) is simple and has a computation complexity of O(N), where N is the number of links in the ring.

Despite its simplicity, the equal allocation policy is not fair because it treats different flows equally regardless of their demands.

4.3 Max–min allocation policy

The third policy is the max–min allocation policy where the flows with demands less than or equal to the per-flow fair share will have their rate allocated first, and the leftover bandwidth will be divided among the other flows which need more than their fair shares.

Let the demands of all flows traversing link i ordered according to their demands such that $b_1 < b_2 < b_3 < \cdots < b_{m_i}$ where m_i is the number of flows traversing link i.

The max–min allocation policy (Fig. 7*b*) will achieve fairness among flows sharing the same link at the expense of a very high computation complexity.

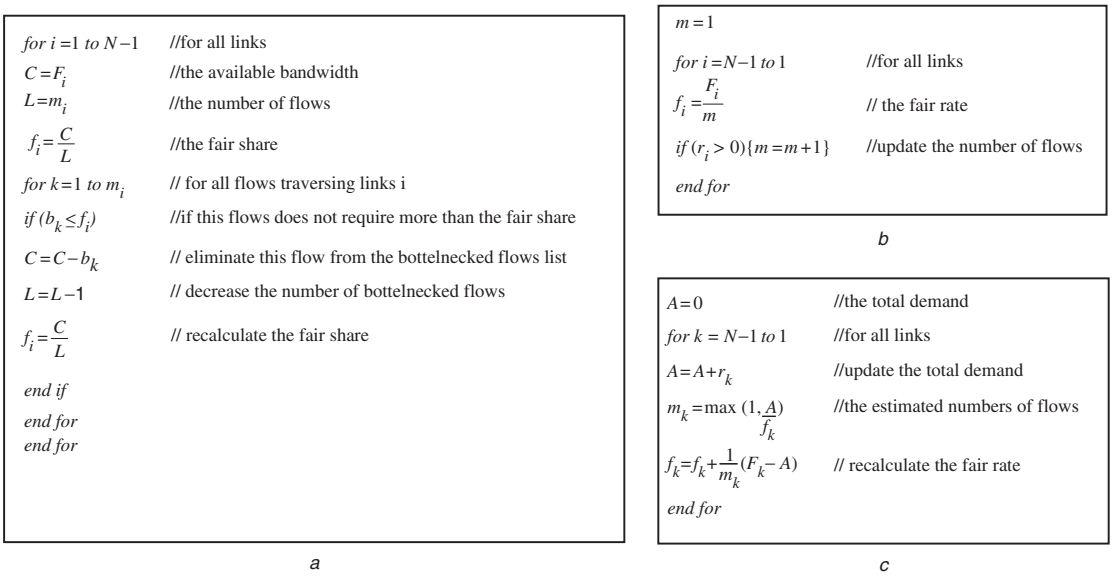

Fig. 7 *Allocation policy*
a The equal allocation policy
b The max–min allocation policy
c The SSA allocation policy

To find the computation complexity of the max–min allocation policy, we consider the worst case where node 1 is sending traffic to all other stations. The per-link fair rate calculation requires a sorting operation with complexity of $O(m_i \log m_i)$. The total complexity can be calculated as follows

$$\sum_{i=1}^{M} m_i \log(m_i), \quad M = N - 1 \tag{3}$$

The number of flows, m_i, is different for each link. Hence, the max–min allocation policy has a computational complexity with a lower bound of $O\left(\frac{M(M-1)}{2}\right)$ [9] which is significantly complex and not scalable.

5 Simple scheduling algorithm

In this Section, we introduce a new allocation policy referred to as the simple scheduling algorithm (SSA). At the end of the nth measurement interval, where $t = nT$, the algorithm first estimates the effective number of flows traversing link i as follows

$$\tilde{m}_i = \frac{A_i(n)}{f_i(n)} \tag{4}$$

where $A_i(n)$ is the sum of flows transmitted from node 1 and traversed link i during the previous interval T, $f_i(n)$ is the per-flow fair rate of the previous interval, and $\tilde{m}_i$ is the effective number of flows traversing link i.

Now, we propose the following formula to estimate the per-flow fair rate

$$f_i(n+1) = f_i(n) + \frac{1}{\tilde{m}_i}(F_i - A_i(n)) \tag{5}$$

The goal is to adjust $f_i(n)$ so that $A_i(n)$ matches the per-source fair rate F_i of link i and $f_i(n)$ converges to the optimal fair rate f_i^*. Note that one of the important features of the SSA algorithm (Fig. 7*c*) is its low computation complexity of $O(N)$.

Theorem: The SSA algorithm generates a sequence that converges to the max–min fair rate.

Proof: See the Appendix.

6 Simulation results

The performance of the SSA algorithm is evaluated by simulations, and the simulation tool we use is the OPNET simulator. Here, we consider two scenarios. First, we consider the scenario illustrated in Fig. 8*a*. The links have the same capacity of 622 Mbits, and each link has a propagation delay of 0.1 ms. All flows are UDP flows and

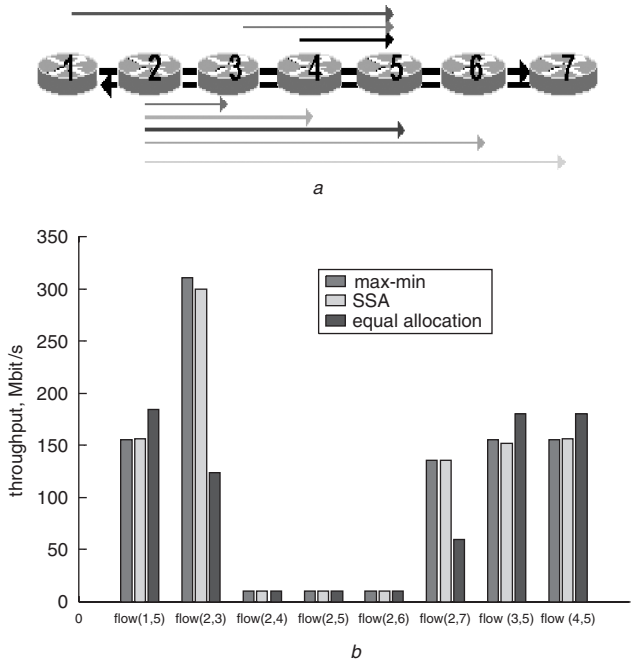

Fig. 8 *Scheduling scenario I*
a Scenario setup
b Policies comparison

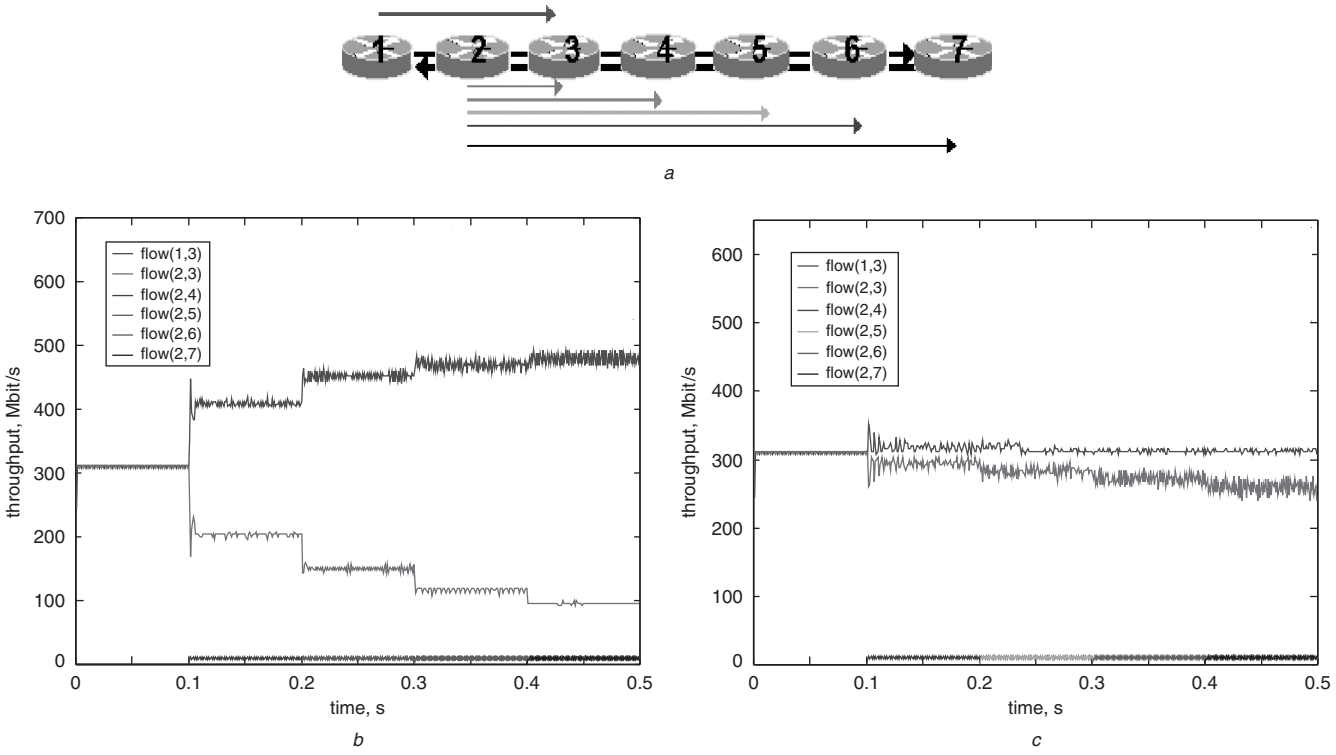

Fig. 9 *Scheduling Scenario II*
a Scenario setup
b Equal allocation
c SSA allocation

start at $t = 0$, flows (1,5), (2,3), (3,5), (4,5) and (2,7) are greedy while flows (2,4), (2,5) and (2,6) are running at a rate equal to 10 Mbits. The measurement time interval was set to $T = 1$ ms.

Using the equal allocation policy, node 2 divides link 4 per-source fair rate ($F_4 = 155.5$ Mbits) by the number of its flows traversing link 4 (in this case 3), thus resulting in a per-flow fair rate of 51.67 Mbits. Flows (2,6) and (2,7) are running at a rate equal to 10 Mbits. Thus, the unused bandwidth at link 4 owing to the equal allocation policy would be reclaimed by other sources. This continues until it is stabilised at link 4 with a fair rate of 180 Mbits and flow (2,5) is only able to get the fair rate equal to 60 Mbits. The same is applied to link 2 where flow (2,3) is only able to get the fair rate equal to 120 Mbits.

Alternatively, using the SSA policy, flows are able to achieve their max–min fair rates.

Figure 8*b* shows the comparison of the allocation policies where the SSA policy is able to achieve the max–min allocation at the same complexity of the equal allocation policy.

Second, we consider the scenario illustrated in Fig. 9*a*. Flows (1,3) and (2,3) are greedy and flows (2,4), (2,5), (2,6) and (2,7) are running at a rate equal to 10 Mbits. The flows, flow (1,3) and flow (2,3), start at $t = 0$, while flow (2,4), flow (2,5), flow (2,6) and flow (2,7) start at time 0.1, 0.2, 0.3 and 0.4 seconds, respectively.

Figure 9*b* shows the unfairness of the equal allocation policy. During the first 0.1 second, when flows (1,3) and (2,3) are active, they equally share the bandwidth of link 2, and both achieve the throughput of 311 Mbits. At the time of 0.1 second, flow (2,4) joins the RPR ring and the allocated bandwidth for each flow on link 2 is decreased to 207.33 Mbits. Since flow (2,4) runs at the rate of 10 Mbits, its throughput is 10 Mbits. Therefore, an unused bandwidth at link 2 of 197.33 Mbits is available for reclaiming by other sources. In this scenario, flow (1,3) reclaims it and its throughput reaches 404.66 Mbits. Similarly, when flows (2,6) and (2,7) join the RPR ring, the throughput of flow (2,3) decreases further and the unused bandwidth is unfairly reclaimed by flow (1,3).

The rate of flow (2,3) decreases at the start of the other flows where the per-flow rate decreases as the number of flows increases despite their low rate demands.

Figure 9*c* shows the performance of the SSA policy where flows are able to achieve their max–min fair rates.

7 Conclusions

In this paper, we have proposed a new traffic shaping scheme for the resilient packet ring to maximise the utilisation and avoid the head-of-line blocking associated with the current RPR traffic shaping scheme. The new scheme uses per-destination queues at the ingress point. Existing bandwidth allocation policies have been investigated and shown to be either inefficient or significantly complex.

An allocation policy, namely, simple scheduling algorithm (SSA), has been proposed, and shown analytically and through simulations to be optimal where the flows achieve their max–min fair rates at a very low computation complexity.

8 Acknowledgments

The authors wish to acknowledge Yanqiu Luo for her help and discussion in enhancing the manuscript. They would also like to thank the anonymous referees for their constructive comments.

9 References

1 IEEE Standard 802.17: Resilient Packet Ring http://ieee802.org/17
2 IEEE Draft P802.17, draft 3.0, Resilient Packet Ring, November 2003
3 Davik, F., Yilmaz, M., Gjessing, S., and Uzun, N.: 'IEEE 802.17 resilient packet ring tutorial', *IEEE Commun. Mag.*, 2004, **42**, (3), pp. 112–118
4 Gambiroza, V., Yuan, P., and Knightly, E.: 'The IEEE 802.17 media access protocol for high-speed metropolitan-area resilient packet rings', *IEEE Network*, 2004, **18**, (3), p. 815
5 Gambiroza, V., Yuan, P., Balzano, L., Liu, Y., Sheafor, S., and Knightly, E.: 'Design, analysis, and implementation of DVSR: a fair, high performance protocol for packet rings', *IEEE/ACM Trans. Netw.*, 2004, **12**, (1), pp. 85–102
6 Alharbi, F., and Ansari, N.: 'Low complexity distributed bandwidth allocation for resilient packet ring networks'. Proc. of IEEE Workshop on High Performance Switching and Routing (HPSR2004), April 2004, pp. 277–281
7 Davik, F., and Gjessing, S.: 'The stability of the resilient packet ring aggressive fairness algorithm'. Proc. 13th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN2004), MillValley, CA, USA, April 24–27 2004, pp. 17–22
8 IEEE Draft P802.17, draft 1.0, Resilient Packet Ring, August 12, 2002
9 Bertsekas, D., and Gallager, R.: 'Data networks' (Prentice-Hall, 1989)
10 Bertsekas, D., and Tsitsiklis, J.N.: 'Parallel and distributed computation' (Prentice-Hall, 1989)

10 Appendix: Proof of convergence

Let i be the bottlenecked link. The number of flows traversing link i is m, where m_i^b is the number of flows bottlenecked elsewhere, and $m_i^u = m - m_i^b$ is the number of flows bottlenecked at link i. Let $r_1^b, r_2^b, \ldots, r_{m_i^b}^b$ be the flows bottlenecked elsewhere, and $r_1, r_2, \ldots, r_{m_i^u}$ are the flows bottlenecked at link i.

At the end of the nth measurement interval ($t = nT$), each node k will use its measured demands to other destinations along with the latest received per-source fair rates from all links and compute the following for each link.

First, the algorithm estimates the effective number of flows traversing link i as

$$\tilde{m} = \frac{\tilde{A}_i(n)}{f_i(n)} \tag{6}$$

where $\tilde{A}_i(n)$ is the sum of flows transmitted from node k and traversed link i during the previous interval, and $f_i(n)$ is the per flow fair rate of the previous interval.

The next advertised fair rate is

$$f_i(n+1) = f_i(n) + \frac{1}{\tilde{m}}(F_i - \tilde{A}_i(n)) \tag{7}$$

Substituting (7) into (6) yields

$$f_i(n+1) = f_i(n)\frac{F_i}{\tilde{A}_i(n)} \tag{8}$$

Define $\alpha(n) = \frac{\tilde{A}_i(n)}{F_i}$ as the load factor, and rewrite (8) as

$$f_i(n+1) = \frac{f_i(n)}{\alpha(n)} \tag{9}$$

According to the load factor value, two cases are considered. First, consider the case where the load factor $\alpha(n)$ is less than one. In this case, the sum of flows traversing link i has a rate less than the link fair rate F_i. According to (9), the per-flow fair rate $f_i(n)$ will increase. If all flows are bottlenecked elsewhere ($m_i^u = 0$), the fair rate has been achieved. Alternatively, if there are some flows bottlenecked at link i ($m_i^u > 0$), the bottlenecked flows will continue to increase their rates until the load factor becomes greater than or equal to one.

Second, consider the case where the load factor $\alpha(n)$ is greater than one. In this case, the sum of flows traversing link i is greater than the link fair rate F_i. According to (9), the per-flow fair rate $f_i(n)$ will decrease and the participating flows will decrease their rates. This will continue until the load factor becomes less than or equal to one.

It is obvious from the above two cases that the load factor oscillates around one and converges to one. Thus, in the following analysis, we assume that the load factor is close to one.

Next, we shall show that the iterative algorithm (7) will generate a sequence of $f_i(n)$ that will converge to the optimal value of the per-flow fair rate $f_i(n) = \frac{F_i - \sum_{j=1}^{m_i^b} r_{bj}}{m_i^u}$.

Note that the iterative equation (7) is in the form of

$$f_i(n+1) = f_i(n) + \lambda\left[\nabla^2 D(f_i(n))\right]^{-1}\nabla D(f_i(n)) \tag{10}$$

That is, the per-flow fair rate is adjusted in the direction of the gradient, where

$$\nabla D(f_i(n)) = F_i - \tilde{A}_i(f_i(n)) \tag{11}$$

Here, λ is a positive step size, and in our case is equal to one, and $\left[\nabla^2 D(f_i(n))\right]^{-1}$ is the inverse of the Hessian.

It is well known that the Newton method (7), where the gradient is scaled by the inverse of the Hessian, typically converges faster than the gradient projection; see [10, pp. 201].

The Hessian $\nabla^2 D(f_i(n)) = \tilde{m}$ is approximated by using two points, the current point of $(\tilde{A}_i(n), f_i(n))$ and the origin (0,0).

Hence, the above iterative equation converges, and the stable value of the per-flow fair rate is detailed as follows:

First, assume that all the flows are bottlenecked at link i. In this case, $m_i^b = 0$ and $m_i^u = m$. All flows are running at the per-flow fair rate $f_i(n)$, and the sum of flows traversing link i is

$$\tilde{A}_i(n) = m f_i(n) \tag{12}$$

Substituting the value of $\tilde{A}_i(n)$ into (8) with a load factor $\alpha(n)$ of one at the steady state yields

$$f_i(n+1) = \frac{F_i}{m} \tag{13}$$

which is the desired value for $f_i(n)$.

Finally, assume that some flows are bottlenecked elsewhere. These flows will have their rates $r_1^b, r_2^b, \ldots, r_{m_i^b}^b$ stabilised, and the allocated bandwidth for these flows is $B = \sum_{j=1}^{m_i^b} r_j^b$.

Since we have a load factor $\alpha(n)$ of one at the steady state, we have

$$\sum_{j=1}^{m_i^u} r_j = F_i - \sum_{j=1}^{m_i^b} r_j^b \tag{14}$$

and

$$\sum_{j=1}^{m_i^u} r_j = m_i^u f_i(n) \tag{15}$$

Substituting (15) into (14) yields

$$f_i(n) = \frac{F_i - \sum_{j=1}^{m_i^b} r_j^b}{m_i^u} \tag{16}$$

Substituting the value of $f_i(n)$ into (9) yields

$$f_i(n+1) = \frac{F_i - \sum_{j=1}^{m_i^b} r_j^b}{m_i^u} \tag{17}$$

which is indeed the desired value for $f_i(n)$ and the proof is complete. ■