Cyclic Queueing Multiple Access (CQMA) for RPR Networks

J.H. Schuringa, G. Remšak, H.R. van As, A. Lila

Vienna University of Technology,
Institute of Communication Networks,
Favoritenstrasse 9/388, A-1040, Vienna, Austria,
e-mail: {Jon.Schuringa, Guenter.Remsak,
Harmen.R.van-As, Arben.Lila}@tuwien.ac.at

Abstract. CQMA is a new resilient packet ring protocol currently proposed to the upcoming 802.17 standard. This paper describes the main mechanisms used in CQMA. These mechanisms are simple and result in a high protocol performance. Two fairness algorithms for CQMA are given, the first computes the maximal theoretical fair rates, is however not well scalable. For larger networks (> 32 stations) a second fairness algorithm is described, which is an approximation of the first algorithm. Not only are the mechanisms simple, the protocol is also simple from an operational point of view: zero parameters are needed for the basic functionality.

1. Introduction

Resilient Packet Ring (RPR) is a technology currently being developed as a new standard for fiber optic rings. The goal is to define an access protocol for use in local, metropolitan and wide area networks for the transfer of data packets at rates scalable to many gigabits per second. Fiber optic rings are widely deployed in metropolitan and wide area networks; however, these topologies are dependent on protocols that are not optimized or scalable to meet the demands of packet-switched networks. This paper presents the proposal “Cyclic Queueing Multiple Access (CQMA)” for the upcoming standard. Although many issues are addressed by the standard, the scope of this paper is to describe the medium access control mechanism of CQMA.

One of the goals of RPR is to have spatial reuse to make efficient and fair use of the available bandwidth on the ring. CQMA achieves this by scheduling the traffic waiting in the transmit queues at all stations in a cyclic manner. A major advantage of the used scheduling method is that it has a proactive congestion control; bottlenecks are detected before they occur. CQMA does not achieve this by backpressure, but schedules the load in each cycle based on the waiting traffic demand. The control information needed to accomplish this, flows in the same direction (i.e. ringlet) as the data flow, which simplifies the protocol in a single ring topology and any configuration of multiple rings.

The remaining of this paper is structured as follows: Section 2 explains the CQMA node structure and the medium access rules. The fairness mechanism, described in Section 3, uses a simple and predictive algorithm, without heuristic thresholds and without the need to measure traffic on the ring. Performance results of CQMA compared with other RPR protocols are given in Section 4. Finally, in Section 5, we give our conclusions.

2. Station Structure and Access Mechanism

The basic structure of a CQMA station is shown in Figure 1; in this figure the structure for only a single ringlet is drawn since both ringlets are independent in CQMA. Packets that arrive at their destination are send to the receive buffers, packets that must be forwarded are immediately forwarded, unless another packet is currently being transmitted. Whenever this is the case, the packet must wait in the transit buffer. Transit traffic always has a higher priority than transmit traffic; the transit buffers are therefore only used for collision avoidance and high priority bypassing. The advantage is not only small MAC end-to-end packet delays, but also small sizes of these buffers, i.e., they can be kept as small as about 1 Maximum Transmission Unit (MTU). CQMA can also operate with only a single transit buffer, but in that case higher priority packets cannot overtake lower priority packets once they are on the ring.

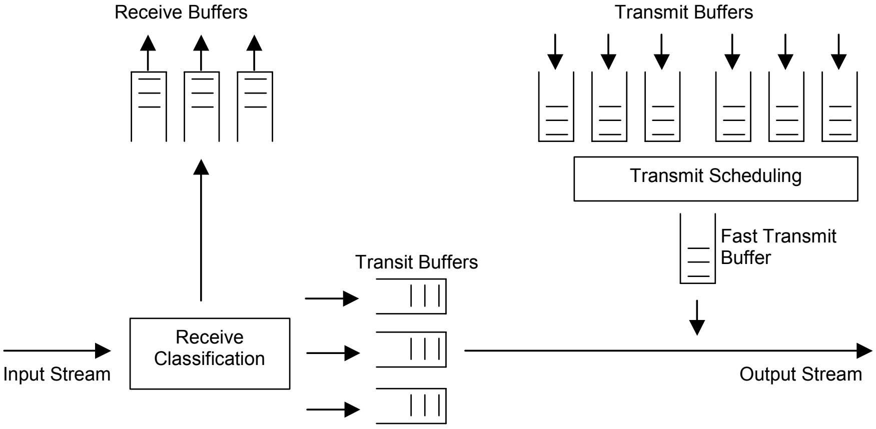


Figure 1. CQMA Station Structure

Fairness is a major issue in RPR, and CQMA handles this in a proactive way: bottlenecks are detected before they occur and the transmit queue scheduler in each station is able to compute the fair rate for each of its flows. This computation is based on the waiting traffic demand; even if the traffic demand completely changes from one cycle to another, the mechanism directly adapts to the new situation.

The medium access rules are as follows: transit buffers are always served before the transmit buffers, and a higher priority transit buffer before a lower priority transit buffer. So packets from the transmit queue can only be served when the transit buffers are empty.

To prevent head of the line blocking, a separate transmit queue exist for each destination on the ringlet, and for each priority. It is the task of the fairness algorithm to determine how many bytes of each flow are allowed in each cycle. Each transmit queue is in one of two possible states: a) the destination station is across on or more bottleneck links (reservation mode), or b) there are no bottlenecks on the ringlet before the destination station (greedy mode). The mode may change between different cycles, but during one cycle the mode remains constant.

Reservation Mode

Whenever the fairness algorithm detects that the total amount of bytes waiting to be transmitted in the next cycle exceeds the available bandwidth on a link, then all destination queues with flows passing this bottleneck link will be in reservation mode. This means that the fairness algorithm calculates fair rates for these queues, and the transmission scheduler ensures that no queue sends more bytes than its fair amount. Some overshooting is permitted, since packet transmission is allowed as long as the limit is not reached. On average, the half of the mean packet size will be send beyond the fair limit.

It is important to note that the traffic over a bottleneck link in a certain cycle must have been reserved in the previous cycle. This is different for traffic in greedy mode, which will be explained next.

Greedy mode

The remaining bandwidth (i.e. available bandwidth minus the total reserved bandwidth) on all links can be used for greedy access. Whenever a station has packets to transmit for which no reservation exists, but the remaining bandwidth on all links allows the transmission of the packet, the station is allowed to do so. It should update its own table with the remaining bandwidth after each non-reserved packet transmission.

The greedy mode helps in keeping the packet delays low for underutilized links, but since it is uncoordinated among stations, it may theoretically lead to unfairness in the current cycle. However, the flows involved will switch to reservation mode in the next cycle and will get a fair rate from that time on. When needed, the unfairness can also be corrected.

The reservation mode and the greedy mode both depend on the results of the fairness operation, which is the topic of the next section.

3. Fairness Operation

Contrary to other current RPR-standard proposals, the fairness algorithm in CQMA is a flow-based fairness algorithm. This section describes two such fairness algorithms that can be used by CQMA. The first one computes the fair rates based on information of all flows, and is therefore able to compute fair rates equal to the maximum theoretical limit. This method has an excellence performance for rings up to about 32 stations (in terms of throughput and delay), for larger rings the amount of information needed for correct operation will grow to sizes that make the protocol slow adapting to traffic changes.

To overcome this, the second fairness algorithm is an approximation of the first algorithm. It uses only a fraction of the information of the first, is however still able to assign fair rates to each flow close to the rates of the first algorithm. The reason why the performance is a little lower is that the second algorithm is not able to fully exploit the possible spatial reuse.

What both algorithms have in common is their ability to simultaneously assign fair rates for multiple priority classes. The next section describes the first fairness algorithm and how it can be used with multiple priority classes.

Fairness Algorithm 1

The fairness algorithm is executed at each station at regular intervals (calculation interval). As can be seen in Figure 2, the input to the algorithm is a local table with the amount of packets waiting in all transmit queues. Two things about the local table are important: a) the information should be as new as possible, and b) all stations should have identical values in their tables at the start of the execution of the algorithm. The easiest way to accomplish this (for rings up to about 32 stations) is to have a single fairness packet circulating on the ringlet, wherein each station writes its own information and reads the information of the other stations. This has to be done before the new calculation interval starts.

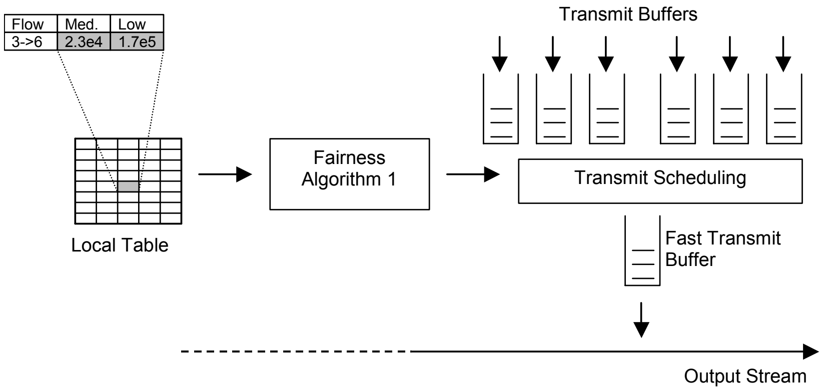


Figure 2. Local flow table is input to the fairness algorithm 1

At the start of a new cycle (calculation interval) each station performs the fairness algorithm. The output of the algorithm is a fair rate for each flow; this information is used by the scheduler to limit the flow for each destination in the calculation interval. Whenever no bottlenecks are detected on the ringlet, or on its segments, each station is able to send up to the still available bandwidth (greedy access) on that segment.

The fair rates are calculated in four steps. In the first step, the high priority traffic (provisioned) over each link is subtracted from the link capacity; we call this the available bandwidth on each link. In the second step, medium priority traffic is being assigned up to the available link capacity multiplied by a factor *f*. This factor controls the minimum amount of medium priority traffic on each link. If, for example, *f* equals 0.9, then medium priority can maximally take up to 90% of the available bandwidth if there is more than 10% low priority. If there is no low priority traffic, then the medium traffic can of course take 100% of the available bandwidth. The third step is to assign low priority up to remaining bandwidth. All bandwidth that still remains after step three can now be assigned to the medium priority traffic.
We now give the pseudo code of the algorithm:

```
MakeFairTop(){
   // step 1: Assign Medium Priority until availableCap * f
   forall Links i
      link[i].remainingCapacity = link[i].availableCap * f
   makeFair1(MedPrio)

   // step 2: Assign low until available capacity
   forall Links i
      link[i].remainingCapacity = link[i].availableCap
   makeFair1(LowPrio)

   // step 3: Assign remaining medium priority to all what is left
   makeFair1(MedPrio)
}
```

The actual fairness algorithm itself is given below in pseudo code. The idea is to take the biggest bottleneck, calculate the fair rate for all flows over that bottleneck and assign those flows. Care should be taken for those flows that want to send less than the fair rate. These flows are handled first. Since this can change the biggest bottleneck link, we search again for the biggest bottleneck. If all flows want to send more than the calculated fair rate, we assign those (allowed variable); set the local table entry for these flows to zero, and update the fairness information on all links involved (updateFairnessBetween). We do this as long as bottlenecks exist. The last step is to copy the entries from the local table to the allowed table and the algorithm finishes.

```
MakeFair1(int prio){
   Do{
      bottleneck = highestBottleneckLink()      // returns the link ID
      if (bottleneck >=0 ) {
         somethingDone   = false
         fairRate        = calcFairRate(bottleneck)

         // first assign those flows that want to send
         // less than the calculated fair rate
         forall flows i over bottleneck {
            if ( flows[i].flow < fairRate) {
               src = flows[i].from
               dst = flows[i].to
               allowed ->plus( src, dst , prio, fairRate)
               tbl     ->set( src, dst , prio, 0)
               updateFairnessBetween(src, dst, fairRate, fairRate)
               somethingDone = true
            }
         }

         // if we assigned flows in the previous code section
         // (i.e. when somethingDone is true) we are done (at
         // least for now with this bottleneck)
         // otherwise assign the fair rates to all flows over the bottleneck
         if (!somethingDone){
            forall flows i over bottleneck {
               src = flows[i].from
               dst = flows[i].to
               value = tbl->get(src, dst, prio)
               if (value>0) {
                  allowed ->plus( src, dst, prio, fairRate)
                  tbl     ->set( src, dst, prio, 0)
                  updateFairnessBetween(src, dst, fairRate,value)
               }
            }
         }
      }
   } while  (bottleneck>0)

   // copy remaining demands (since they are not involved in a bottleneck)
   for (i=0;i<nrNodes;i++) for (j=0;j<nrNodes;j++)
         allowed->plus(i,j,prio,tbl->get(i,j,prio));
}
```

Two functions need further explanation:

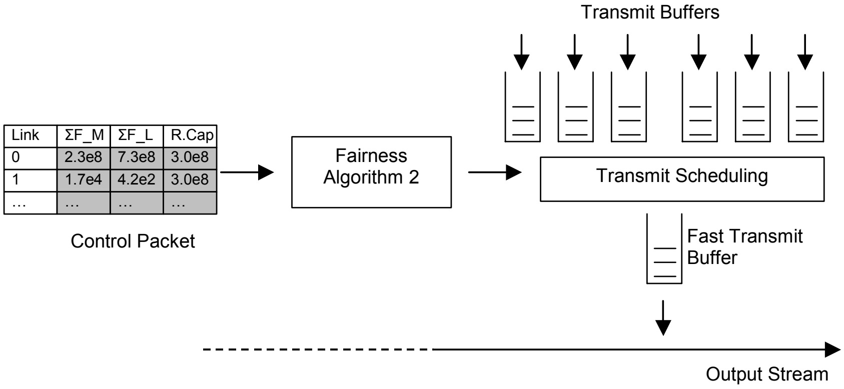
- calcFairRate(*bottleneck*)

  This function returns the available bandwidth on the bottleneck link divided by the number of source destination flows over that link.
- updateFairnessBetween(*src*, *dst*, *rate1*, *rate2*)

  This function updates the remaining capacity, the total traffic demand and the number of flows on all links between source (*src*) and destination (*dst*). The remaining capacity is decreased by *rate1*, the total traffic demand by *rate2*.

Fairness Algorithm 2

Although algorithm 1 achieves the maximum theoretical fair rates, the amount of needed information gets too much for large networks. Algorithm 2 is an approximation of the first, but is scalable and still gets very good results.



Link	ΣF_M	ΣF_L	R.Cap
0	2.3e8	7.3e8	3.0e8
1	1.7e4	4.2e2	3.0e8
…	…	…	…

Figure 3. The control packet is input to the fairness algorithm 2

The control packet contains three arrays each of size N (the number of stations in the ring), two for the total traffic demand on all links for medium and low priority and one array for the remaining capacity (total capacity minus provisioned traffic) on all links.

The operation is as follows: One designated station (e.g. the one with the lowest MAC address) has a timer for each ringlet that fires every calculation interval. At this event the designated station creates a fairness control packet, initializes it to zero and starts the first round of this packet. In this “information gathering” round each station writes in the remaining capacity field, the amount of bytes that are available for its outgoing link in the next cycle. Additionally it also adds its own flows (information comes again from the waiting traffic demand) to the sum of all flows on all links. Once the control packet arrives back at the designated station, the control packet starts its second round where each station performs algorithm 2 and immediately can start sending its fair share. The control packet is taken from the ringlet at the time it returns for the second time at the designated station.

Since the control packet is relative small in size, it produces a small overhead even for short calculation intervals. It is possible to automatically set the calculation interval to a safe minimal value (at least 2 times the maximum round trip time), in which case the CQMA protocol needs *zero* parameters for the basic protocol operation.

Pseudo code of algorithm 2:

```
MakeFair2(int prio){
  Do{
    bottleneck = highestBottleneckLink()      // returns the link ID
    if (bottleneck >=0 ) {
      fairRate        = calcFairRate(bottleneck)

      forall flows i that this node sends over bottleneck {
          dest                = flow[i].to
          oldValue            = myDemand[prio][dest]
          newValue            = oldValue / fairRate
          myDemand[prio][dest]= 0
          myAllow[prio][dest] = newValue

          forall links k between this node and destination dest
            pck->table.available[k]        -= newValue
            pck->table.demand[prio][k]     -= oldValue
          }
      }
    }
  } while  (bottleneck>0)
```

```
   // add all traffic not involved in a bottleneck to myAllow
   myAllow += myDemand
}
```

The function “calcFairRate” in algorithm 2 is different from the one in algorithm 1; this function now returns a rate proportional to the total traffic demand (available bandwidth divided by the total traffic demand).

The performance of both algorithms, compared with other RPR protocols is given in the next section.

4. Performance Evaluation

We present one asymmetric traffic scenario for a double 622Mbit/s ring with 16 stations. All stations send saturated low priority traffic to all other stations. The exceptions are source stations 2 and 3, who only send to stations 13 and 9 respectively. We compare both presented algorithms and 2 other RPR protocols: Alladin and Darwin [1]. All results are obtained using simulation models implemented in IKNsim, our institute’s simulation tool.

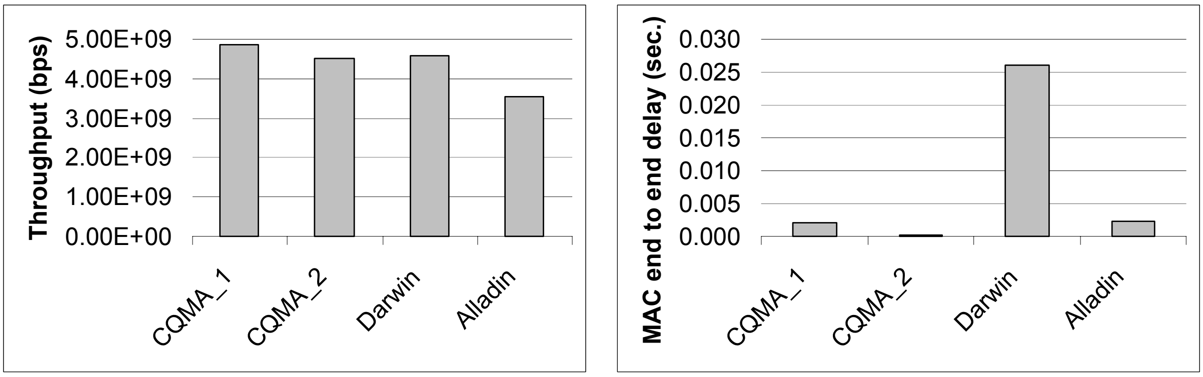


Figure 4. Total throughput and MAC end to end delay

Figure 4 shows that CQMA with algorithm 1, indeed has an higher throughput than with algorithm 2. Darwin has a slightly higher throughput than CQMA_2, but Darwin has a much higher delay than all other protocols. The main reason for this is that Darwin uses the transit queues for scheduling. Interesting is the very low delay for CQMA_2, caused by the much smaller calculation intervals that are possible with fairness algorithm 2.

In Figure 5, throughput per source station, the difference between both fairness algorithms is very well displayed; CQMA_1 can use all remaining bandwidth.

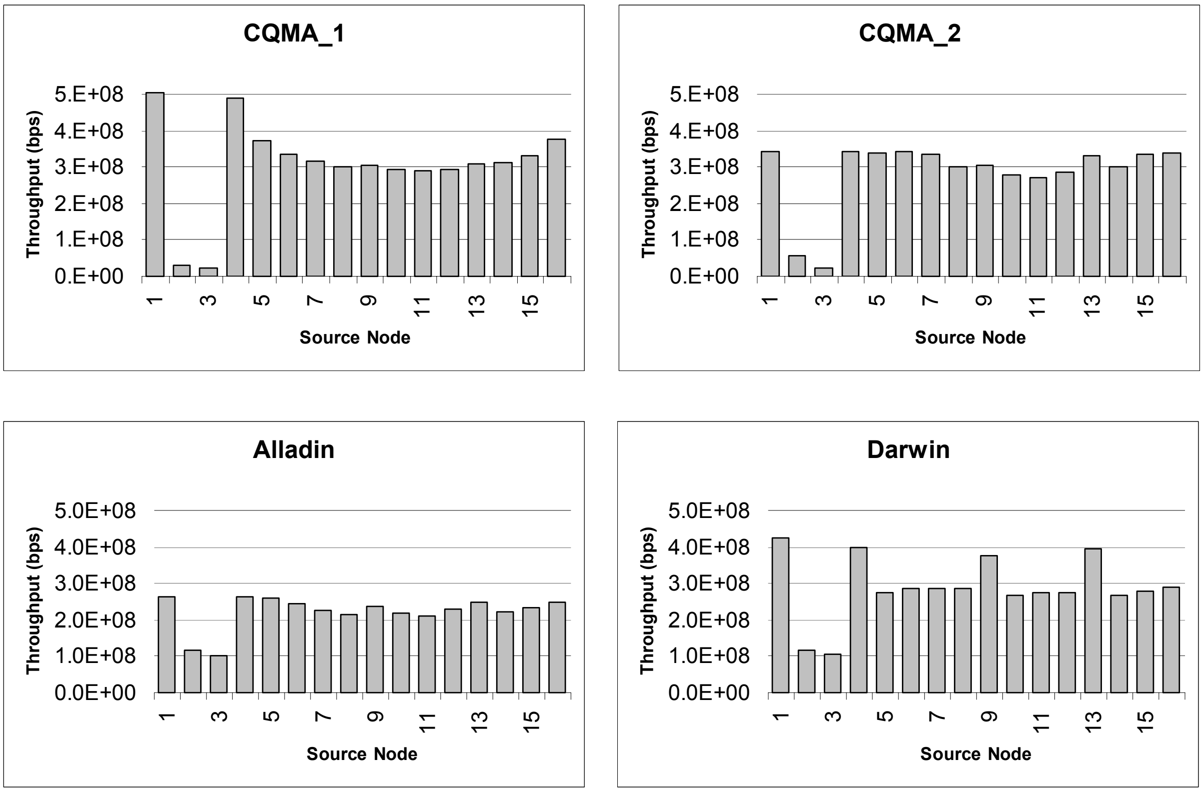


Figure 5. Throughput per source station

5. Conclusion

CQMA has a very high performance with both fairness algorithms. Fairness algorithm 1 provides a throughput equal to the theoretical value, but needs a relative large amount of information that grows nonlinear with the network size. Algorithm 2 is a scalable approximation, which uses much less information and therefore can work with much smaller cycles resulting in extreme low MAC end to end delays.

The medium access mechanism is simple and straightforward and can handle three priority classes: high (provisioned), medium and low traffic, of which the last two are separately assigned a fair rate in case of congestion. Stations wanting to transmit over non-congested links may send directly in greedy mode, to keep delays to a minimum.

References

[1] Unofficial 802.17 draft proposals,
http://grouper.ieee.org/groups/802/17/documents.htm