

# Interleaved DQRAP with Global TQ

by

Chien-Ting Wu

Graham Campbell

Computer Science Dept. - Illinois Institute of Technology

DQRAP Research Group Report 944<sup>1</sup>

**Abstract:** This document describes the deployment of DQRAP in those environments where  $a > 0.5$  where  $a$  is the ratio of propagation delay to frame transmission time in a network. The conventional solution is to establish an interleaving factor equal to or greater than  $2a+1$  and to then operate  $2a+1$  protocol engines in parallel. Arriving packets compete in a given slot and stay with that slot till transmission is complete. The algorithm presented in this paper utilizes the usual interleaving factor of  $2a+1$  but after contention is resolved all packets awaiting transmission enter a common global queue. Algorithms are presented accompanied by diagrams illustrating the operation of Global TQ DQRAP.

## 1 Introduction

Massey [1] describes interleaving as applied to the tree protocols but the concept is generic and can be applied to most any protocol to compensate for the  $a > 0.5$  problem. Interleaving solves the utilization problem in that utilization in a network employing interleaving will be the same as if the protocol is deployed in an  $a \leq 0.5$  network. There is an increase in delay due to the extra propagation, i.e., stations awaiting feedback, but this is compounded in what can be described as the standard interleaving solution. The problem is that statistically multiple packets will arrive in a single slot while an adjacent slot remains unused. Thus the average delay is higher than what is theoretically possible. Global DQRAP addresses this problem. Familiarity with DQRAP as described by Xu and Campbell in [2]

## 2 Global TQ

**2.1 Channel Model** In an interleaving environment, the channel is divided into slots of fixed length. Each slot consists of  $m$  CMS' followed by a single data slot. If the value of  $a$  is  $a$ , then the interleaving factor is  $n \geq 2*a + 1$ . When a station transmits in a slot, it can expect that the slot will be received after  $n - 1$  slot times. Figure 1 shows a channel model with  $a = 2$  and interleaving factor 5. In this figure, 4 slots are in the air, and one slot is in the node's buffer.

---

1. Manuscript prepared Dec 14, 1994

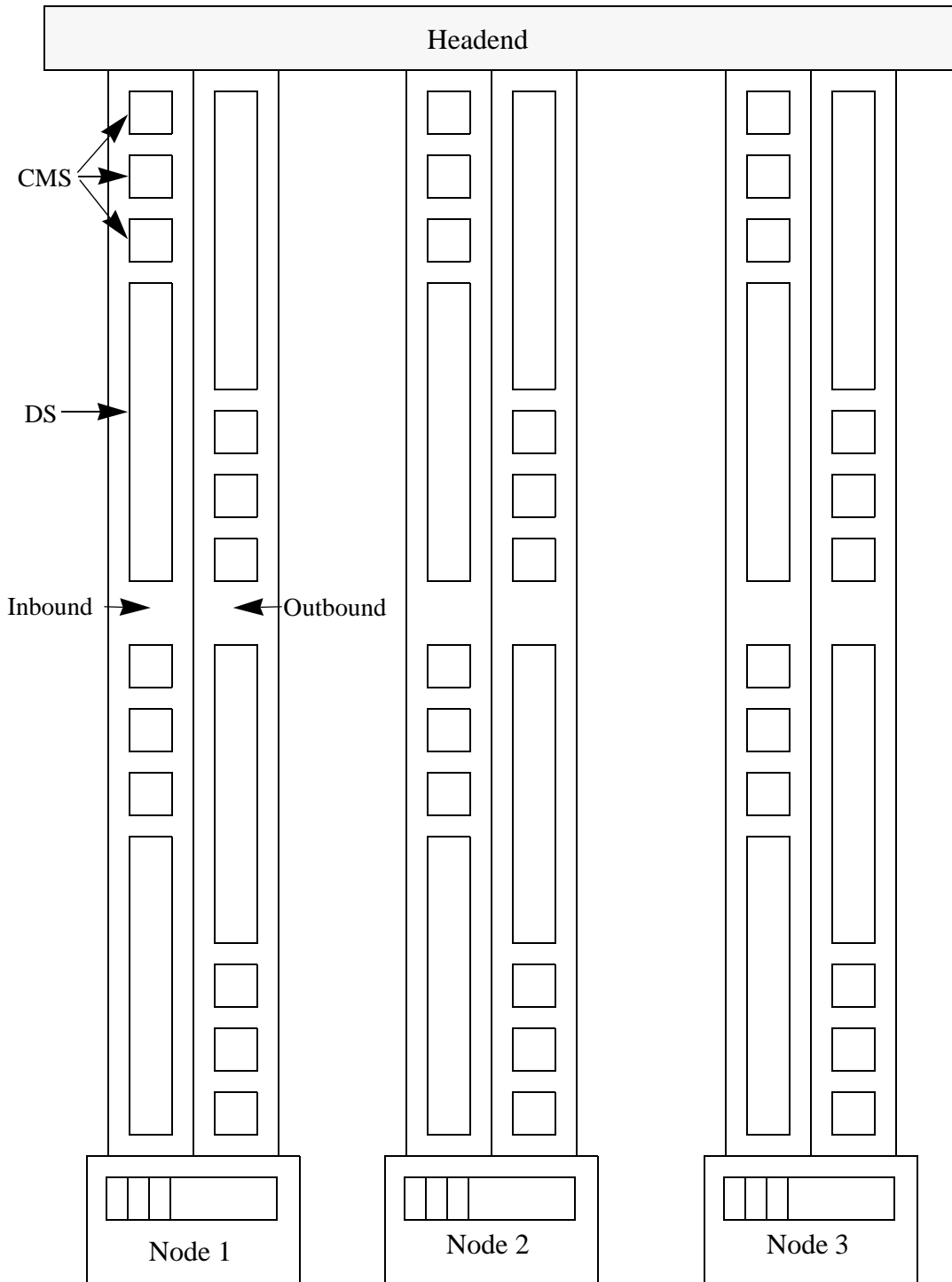


Figure 1 Interleaving Channel Model

**2.2 Independent DQRAP Model** The straightforward method to apply DQRAP in an interleaving environment with interleaving factor  $n$  is to put  $n$  independent DQRAP engines into this system. Each DQRAP engine maintains its own  $TQ_i$  and  $RQ_i$ ,  $i = 1 \dots n$ , and will be active once in each cycle. One cycle consists of  $n$  slot times. In every cycle, each DQRAP engine  $i$  applies QDR to modify  $TQ_i$  and  $RQ_i$  based on the received CMS' which were sent in the last cycle, and then applies RTR and DTR according to the updated  $TQ_i$  and  $RQ_i$ . The active sequence in one cycle starts from engine 1, engine 2, and so on, to engine  $n$ .

The working model is shown in Figure 2. In this example the interleaving factor is 5, and 5 independent DQRAP engines are presented. Each engine has its own resolution queue and transmission queue. Note that there is only one arrival queue (AQ) for the system. Each DQRAP engine will check this common AQ when it applies RTR and DTR to send requests and data. Once an arrived packet has been processed by one DQRAP engine to send request and data, this packet will be transmitted from the same DQRAP engine.

Figure 3 to Figure 12 show how the independent DQRAP model operates in the environment with interleaving factor 3. In this example, the number of CMS' is 2. Figure 3 shows that packet 1 and 2 have arrived and DQRAP engine 1 is active. Since  $RQ_1$  and  $TQ_1$  are zero, requests are sent into CMS' and data are sent into data slot. In Figure 4, DQRAP engine 2 is active, and two more packets arrive. Again, these two packets send requests and data into the CMS' and data slot. DQRAP engine 3 becomes active in Figure 5, packet 5 arrives and sends a request and data into a CMS and data slot. The event in Figure 6 is that DQRAP engine 1 receives CMS feedback. Since the requests of packet 1 and 2 collided in the second CMS, packet 1 and 2 enter  $RQ_1$ , and then send requests again. Packet 6 arrives, but  $RQ_1$  is greater than zero, thus it can send neither request nor data at this moment. In Figure 7, DQRAP engine 2 receives the feedback from packets 3 and 4. Packets 3 and 4 sent requests in the first CMS and second CMS respectively, so packet 3 enters the first entry of  $TQ_2$ , and packet 4 enters the second entry of  $TQ_2$ . Packet 3 sends data in a data slot following the DTR and Packet 6 sends a request in a CMS following the RTR. In Figure 8, there are no new arrivals,  $RQ_3$  and  $TQ_3$  are all equal to zero, so nothing happens in this slot. In Figure 9, DQRAP engine 1 receives the feedback from packet 1 and packet 2. Packets 2 and 1 sent requests into the first and second CMS respectively, therefore they enter the first and second entries of  $TQ_1$  respectively. Packet 2 sends data in the data slot. In Figure 10, packet 4 is the first entry of  $TQ_2$  and sends data in the data slot, packet 6 sent a request in the first CMS without a collision and enters the second entry of  $TQ_2$ . In Figure 11, DQRAP engine 3 is active, but there are no new arrivals and no packets in  $TQ_3$  and  $RQ_3$ , so this slot is idle. In Figure 12, DQRAP engine 1 sends packet 1 into a data slot.

In Figures 8 and 11, we find that the data slot is idle but there are packets waiting transmission in the other transmission queues. The values of  $TQ_i$  and  $RQ_i$  are maintained by all of the nodes, so even though logically the DQRAP engines are independent, their  $TQ$  and  $RQ$  values are known to each other. This suggests that when the current DQRAP engine finds its  $TQ$  is zero, it can help other DQRAP engines by transmitting packets waiting in their  $TQ$ . Figure 13 and Figure 14 show the idea. In Figure 13, DQRAP engine 3 is active,  $TQ_3$  and  $RQ_3$  are zero, so it can move

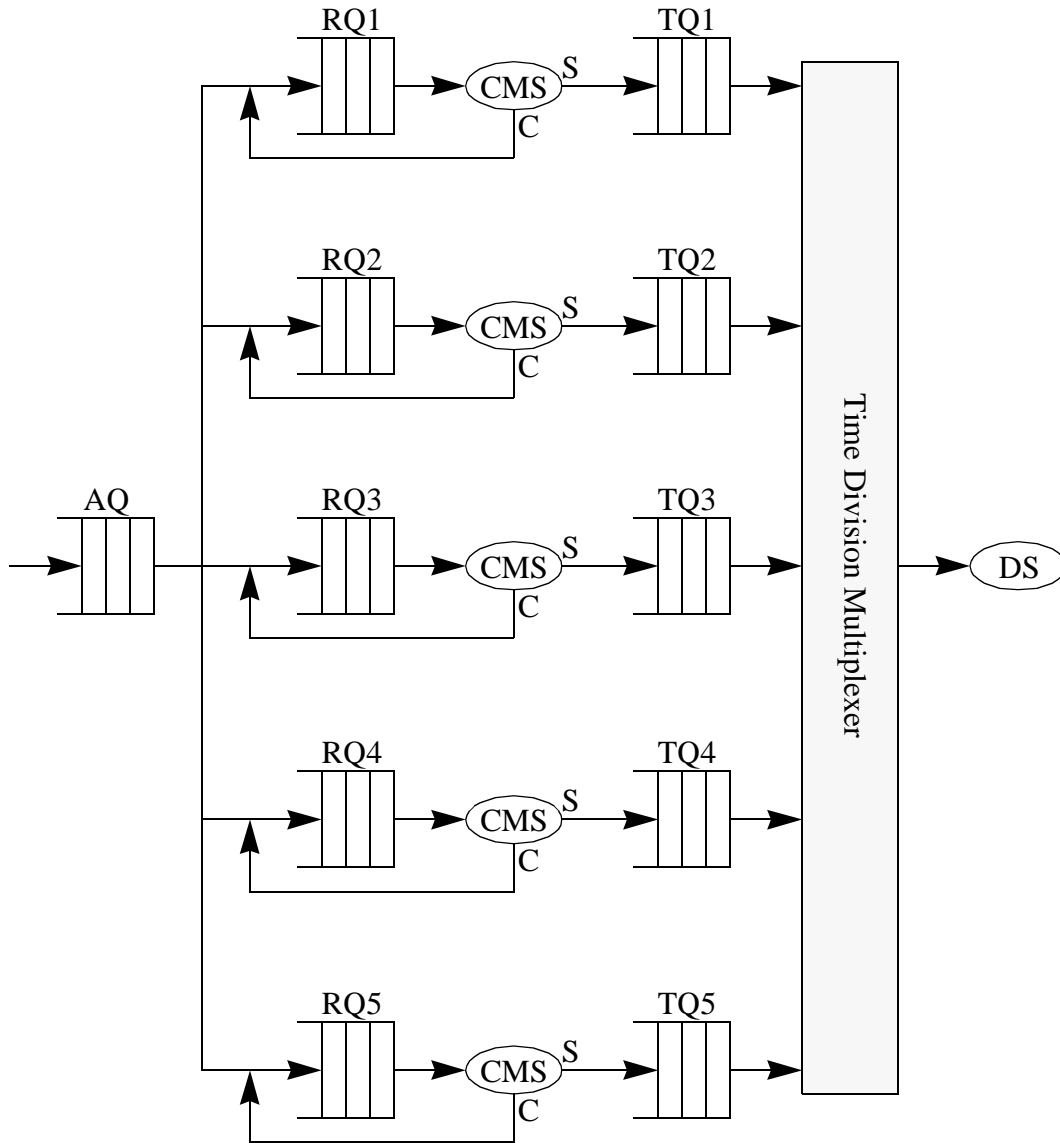


Figure 2 Independent DQRAP Model with Interleaving Factor 5

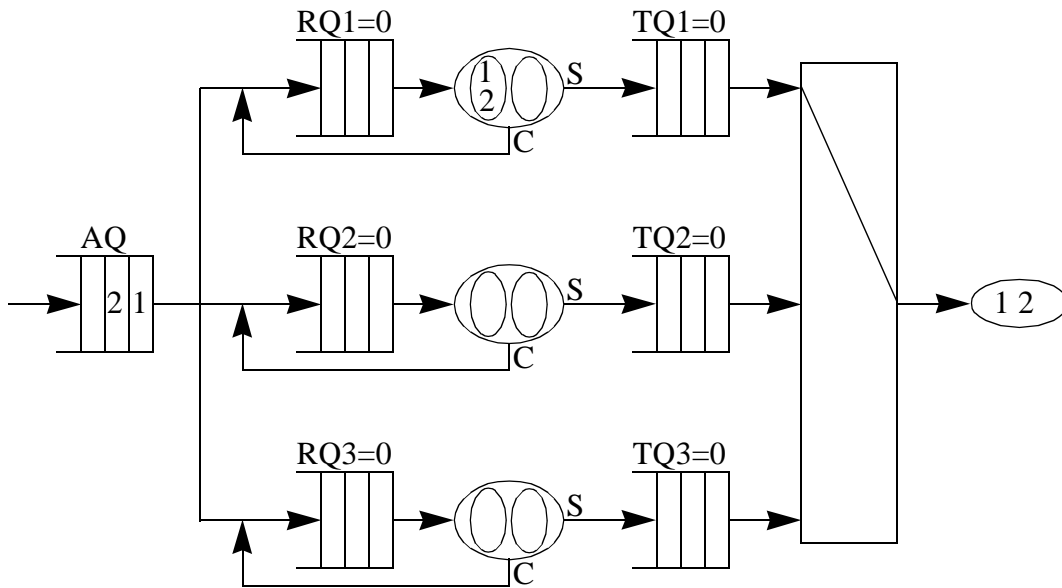


Figure 3 Separate TQ Example: Engine 1 Active in the First Cycle

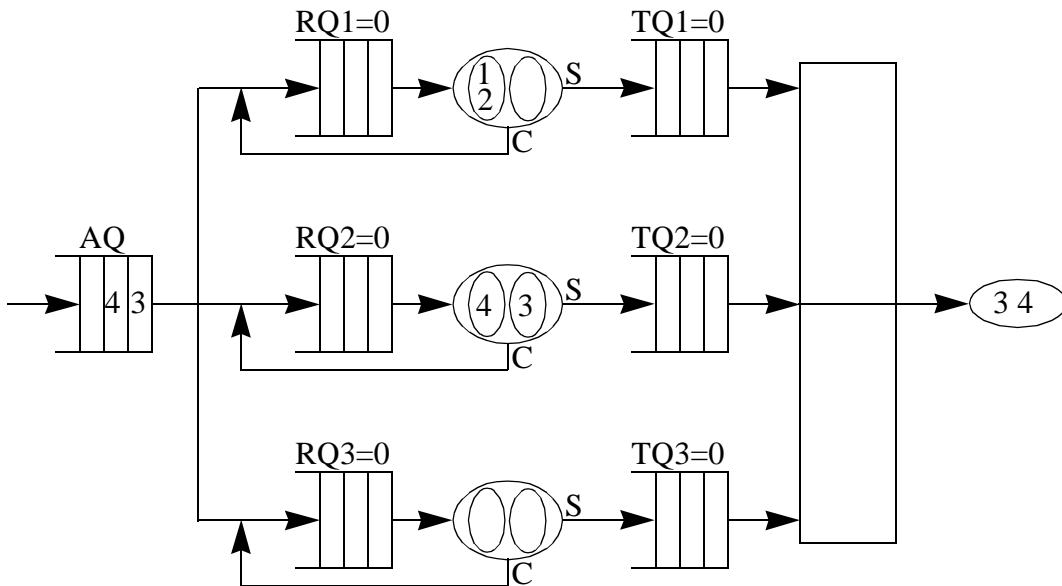


Figure 4 Separate TQ Example: Engine 2 Active in the First Cycle

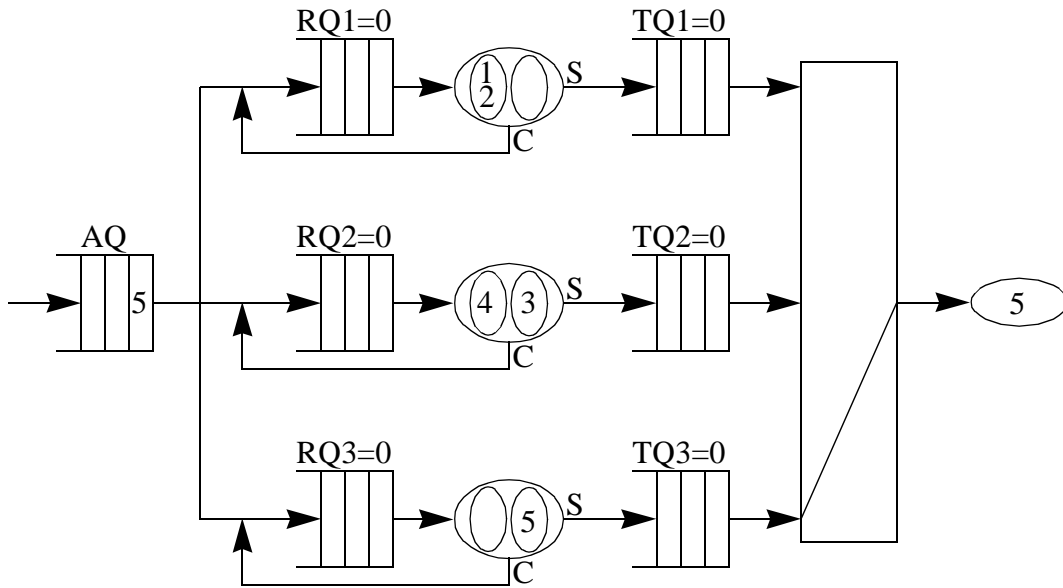


Figure 5 Separate TQ Example: Engine 3 Active in the First Cycle

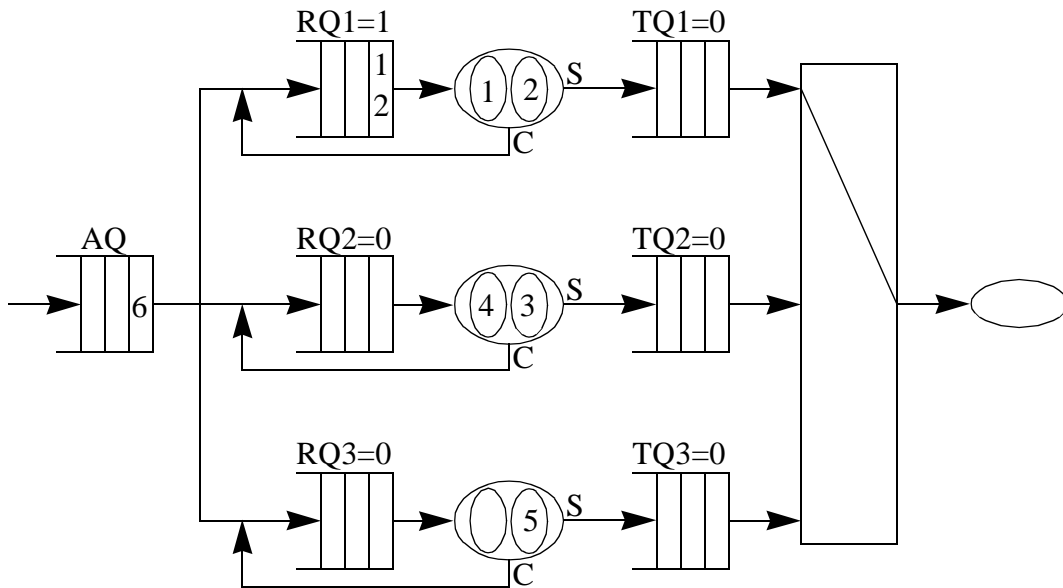


Figure 6 Separate TQ Example: Engine 1 Active in the Second Cycle

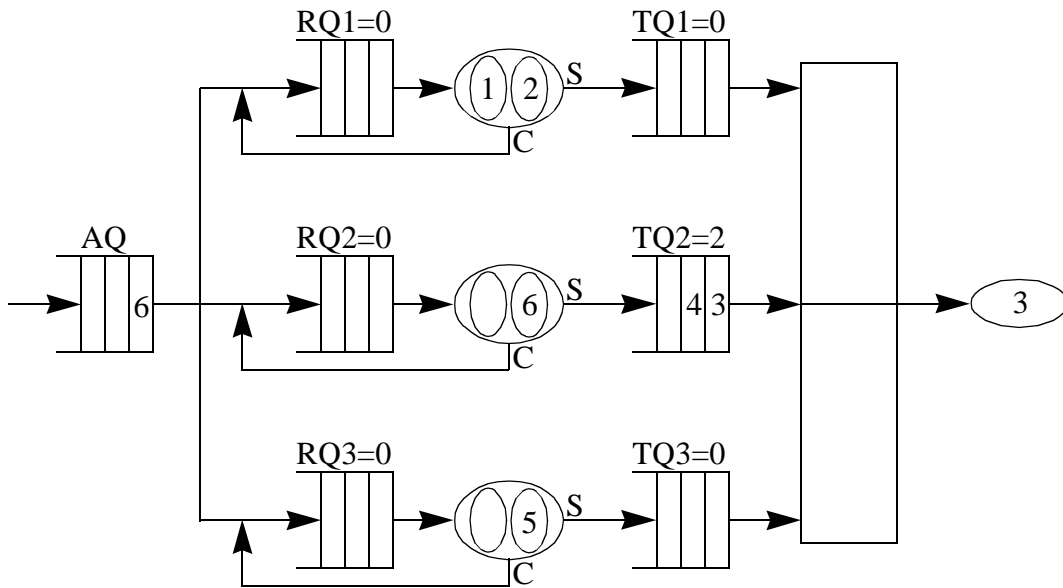


Figure 7 Separate TQ Example: Engine 2 Active in the Second Cycle

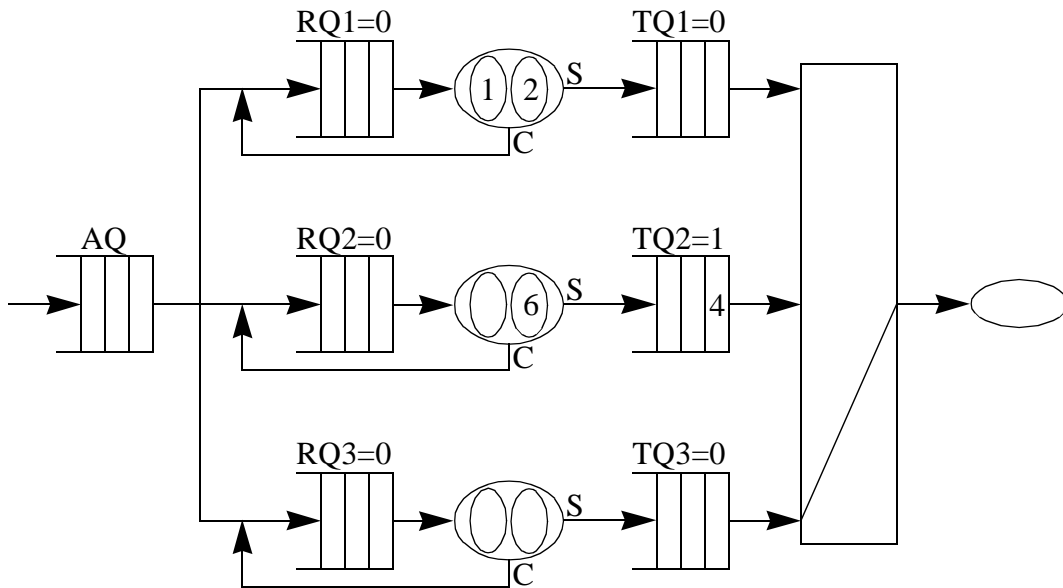


Figure 8 Separate TQ Example: Engine 3 Active in the Second Cycle

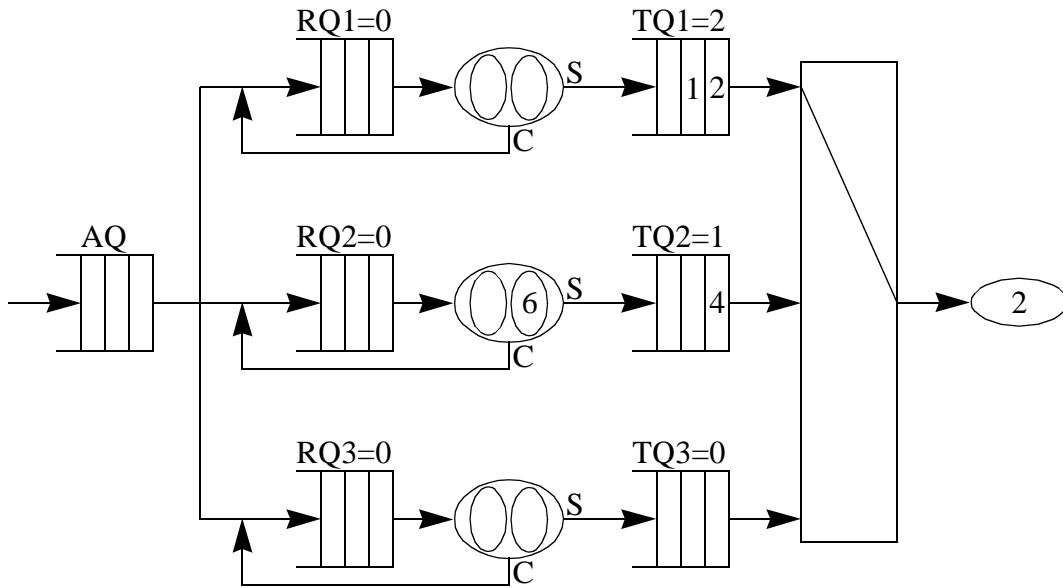


Figure 9 Separate TQ Example: Engine 1 Active in the Third Cycle

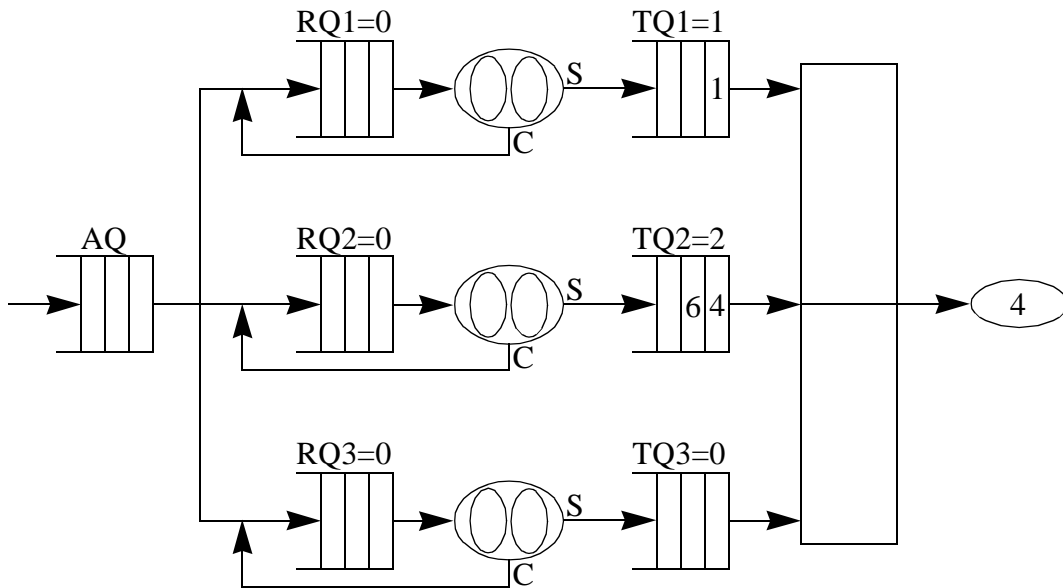


Figure 10 Separate TQ Example: Engine 2 Active in the Third Cycle

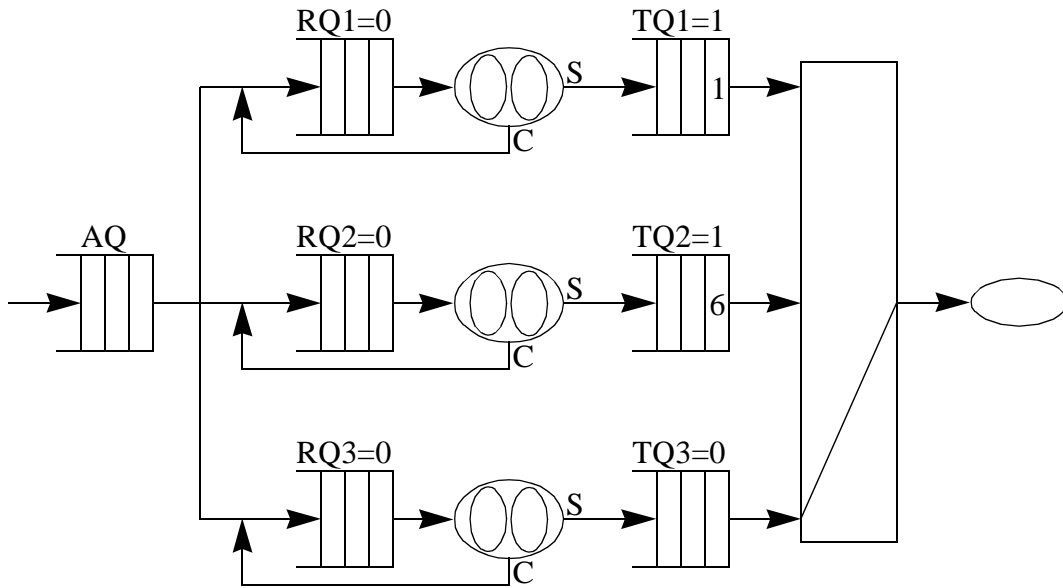


Figure 11 Separate TQ Example: Engine 3 Active in the Third Cycle

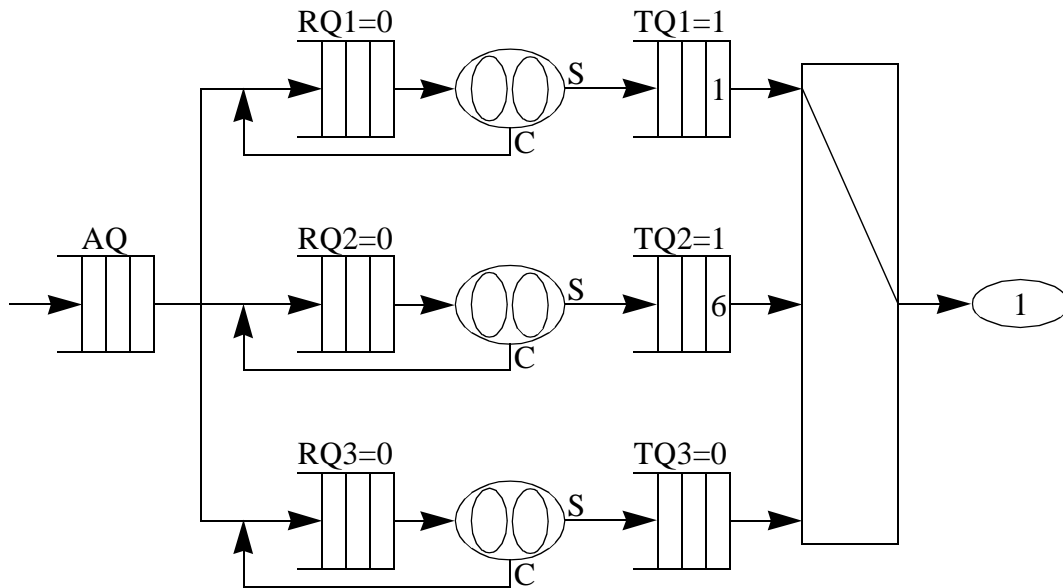


Figure 12 Separate TQ Example: Engine 1 Active in the Fourth Cycle

packet 4 from TQ2 to TQ3 and transmit this packet. In Figure 14, DQRAP engine 3 can move either packet 1 from TQ1 or packet 6 from TQ2 to TQ3. This reduces the average delay for the system. From this observation we conclude that a single TQ should do the job, but the number of RQs needed is still the same as the interleaving factor because a transmitted request must wait (interleaving factor - 1) slot time to receive. The difference between the independent DQRAP model and single TQ model is based on the number of TQs used, thus the independent DQRAP model is referred to as the separate TQ model, and the single TQ model is referred to as the global TQ model in the rest of the dissertation. The global TQ model and protocol are described in the next section.

**2.3 Global TQ protocol** Figure 15 shows a global TQ model with interleaving factor 5. In the global TQ model, the requests sent into the CMS' still have to wait (interleaving factor - 1) slot times to obtain feedback. When the collision has been resolved, they enter into a common TQ. Therefore, as long as the TQ is non-empty a data slot will not be wasted.

In the global TQ model with interleaving factor  $n$ , each node maintains  $n$  resolution queues,  $RQ_i$ ,  $i = 1 \dots n$ , and one single TQ. In order to differentiate whether the received packet is sent when TQ and RQ equal to 0, each node maintains  $Prev\_TQ_i$ ,  $i = 1 \dots n$ . Let  $F_k$ ,  $k = 1, 2, \dots, m$ , denote feedback from the  $k_{th}$  CMS. The values of  $F_k$  can be E:empty, S:single, or C:collide. If a station transmitted a request in a CMS, it will enter the TQ or RQ in the order of the CMS numbered from 1 to  $m$ . The global TQ protocol is stated as follows:

```
Global_TQ_Protocol()
{
    int i, group;
    TQ = 0;
    for (i = 1; i <= interleaving_factor; i++) {
        RQi = 0;
        Prev_TQi = 0;
    }
    group = 1;
    while(TRUE) {
        Detect_CMS_Feedback();
        Global_TQ_QDR(group);
        Global_TQ_RTR(group);
        Global_TQ_DTR(group);
        group = (group % interleaving_factor) + 1;
    }
}
```

```
Global_TQ_QDR(int group)
{
```

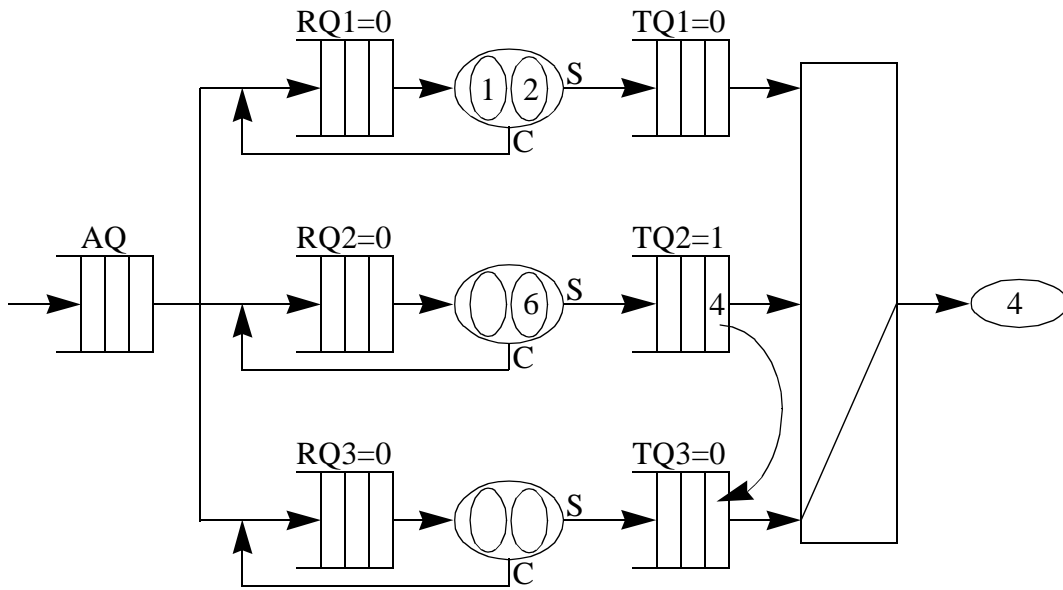


Figure 13 Move Packet 4 from TQ2 to TQ3

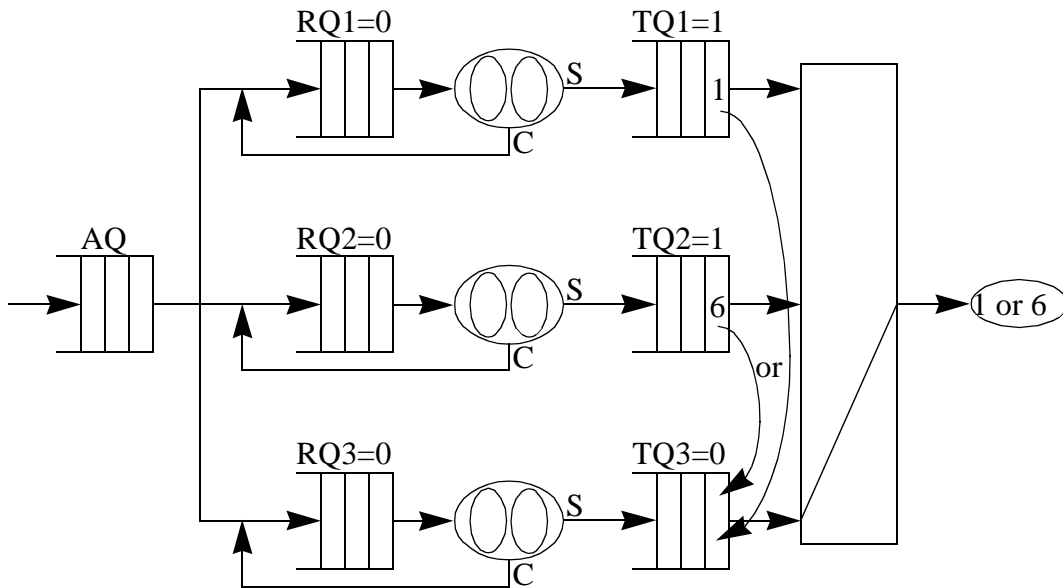


Figure 14 Move Packet 1 from TQ1 or Packet 6 from TQ2 to TQ3

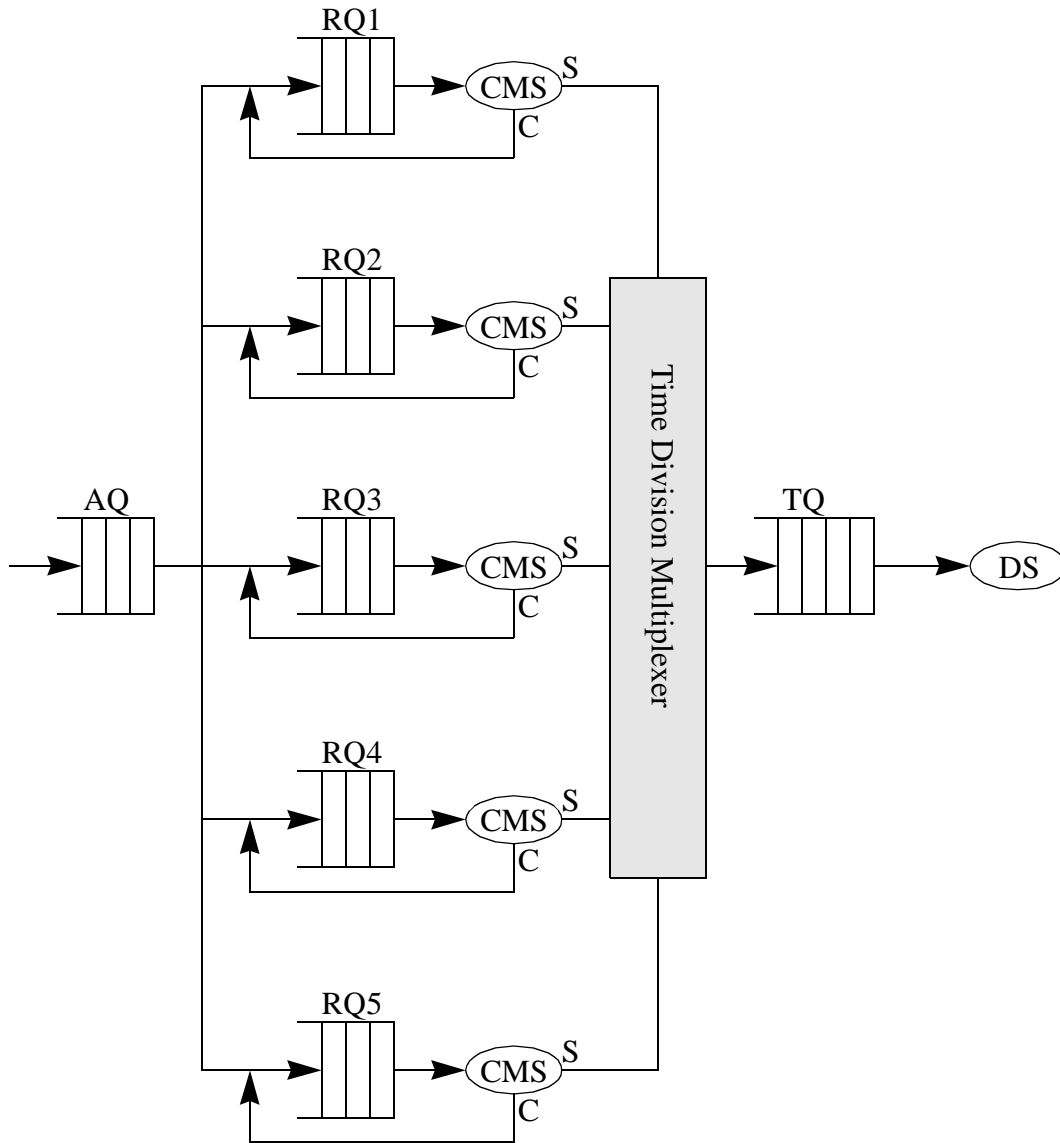


Figure 15 Global TQ Model with Interleaving Factor 5

```

int k, n_request;
for (n_request = 0, k = 1; k <= m; k++)
    if (Fk == S)
        n_request++;
    else if (Fk == C)
        n_request += 2;
for (k = 1; k <= m; k++) {
    if (Fk == S) {
        if (Prev_TQgroup == 0 && n_request == 1)
            /* Immediate access */
        else
            TQ++;
    }
    else if (Fk == C)
        RQgroup++;
}
Prev_TQgroup = TQ;
}

```

```

Global_TQ_RTR(int group)
{
    if (RQgroup > 0) {
        if (the node has packet on the first entry of RQgroup)
            send request into CMS
    }
    else {
        if (the node has new arrival packet)
            send request into CMS
    }
}

```

```

Global_TQ_DTR(int group)
{
    if (TQ > 0) {
        if (the node has the packet queued in the first entry of TQ)
            send data into DS
        TQ--;
    }
    else if (RQgroup == 0) {
        if (the node has new arrival packet)
            send data into DS
    }
}

```

```

    }
    if (RQgroup > 0)
        RQgroup--;
}

```

Figure 16 to Figure 23 use the same example presented in the separate TQ example to demonstrate the operation of global TQ. Figure 16 shows that packets 1 and 2 arrive and RQ1 is active. Since RQ1 and TQ are zero, requests are sent in CMS' and data are sent in a data slot. In Figure 17, RQ2 is active, and two more packets arrive. Again, these two packets send requests and data in CMS' and data slot. In Figure 18, packet 5 arrives and sends a request and data in a CMS and data slot. In Figure 19, the feedback of requests sent by packets 1 and 2 are received. Since the requests of packet 1 and 2 are collided in the second CMS, packets 1 and 2 enter RQ1, and then send requests again. Packet 6 arrives, but according to RTR and DTR, it can send neither request nor data at this moment. In Figure 20, the requests from packet 3 and 4 are received without a collision, so packets 3 and 4 enter the TQ. Since RQ2 is zero and TQ is 2, packet 6 can only send a request in a CMS. In Figure 21, no new arrivals, but packet 4 is in the first entry of TQ, so packet 4 is sent in a data slot. In Figure 22, the requests from packet 1 and 2 are received without collision, so they enter into TQ, and packet 2 is sent in a data slot. In Figure 23, packet 1 is the first entry of TQ, and the request from 6 is received without collision, so packet 6 enters the second entry of TQ. From this example, we see that packet 1 is transmitted at 8th slot, two slots sooner when compared to the separate TQ example.

#### 4 Improved Global TQ Protocol

The global TQ model is derived from the separate TQ model by merging multiple TQs into a single TQ. Now, the question is: can we apply the same idea to multiple RQs to further improve the performance? Let's look at the situation in Figure 24. In Figure 24, RQ2 is active and is equal to zero. If there are new arrivals, the new arrival packets will send request into CMS', otherwise the CMS will be empty in this slot. However, in RQ1, packets 2 and 6 just sent requests into CMS in the previous slot and are waiting for the feedback. Packets 3 and 5 have to wait until the next cycle to send a request. If we can determine that the CMS' will be empty in this slot, we could move packets 3 and 5 from RQ1 to RQ2, and let them utilize the current CMS to send requests. The contention would be resolved earlier and the delay decreased. But, the problem is that there is no way for a node to know whether there are new arrivals in the other nodes. For example, in Figure 24, only the nodes holding packets 7, 8 and 9 know the AQ is not empty, the rest of the nodes do not have this information. However, we still can move the first entry of RQ1 to RQ2 and block the new arrivals, if any. This is shown in Figure 25. This method will reduce the delay for packets in the first entry of RQ1. So, if AQ is empty, the average delay is reduced slightly. If AQ is not empty, we still do not lose the performance in average delay, because the reduced delay in packets 3 and 5 could offset the longer delay in packets 7, 8 and 9. This also makes this protocol closer to a FIFO system. Following is the modified QDR algorithm:

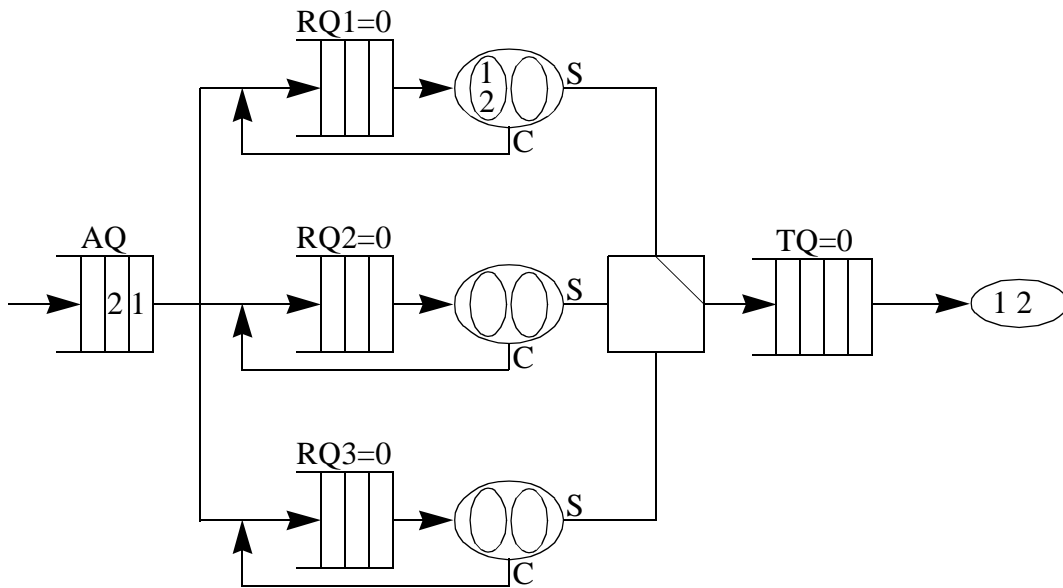


Figure 16 Global TQ Example: RQ1 Active in the First Cycle

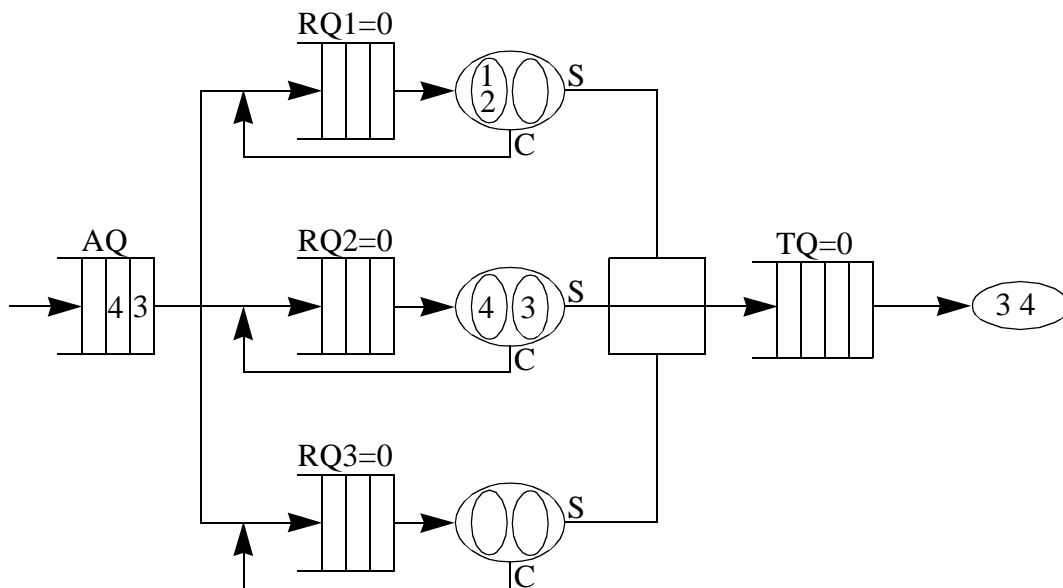


Figure 17 Global TQ Example: RQ2 Active in the First Cycle

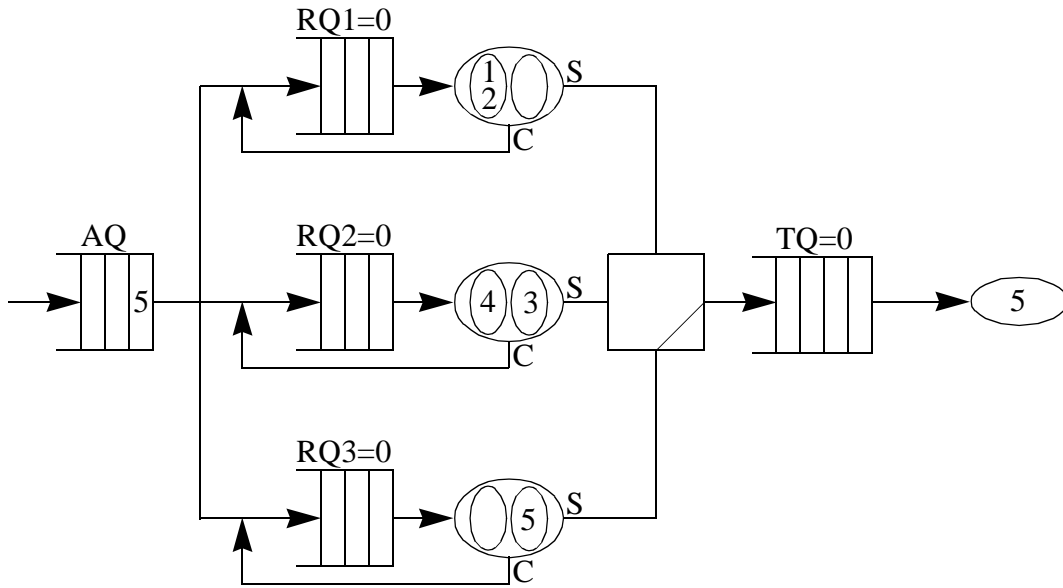


Figure 18 Global TQ Example: RQ3 Active in the First Cycle

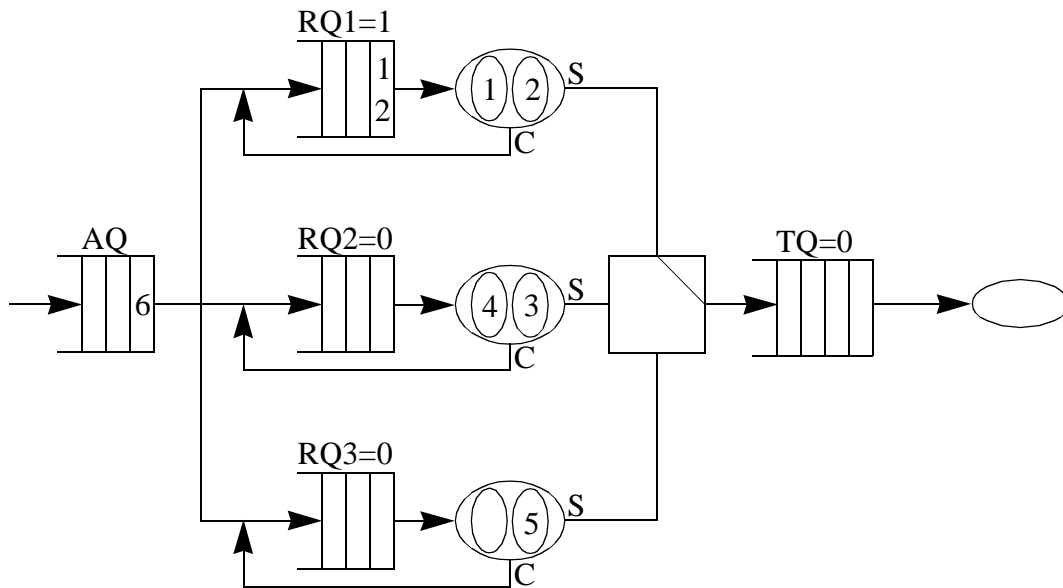


Figure 19 Global TQ Example: RQ1 Active in the Second Cycle

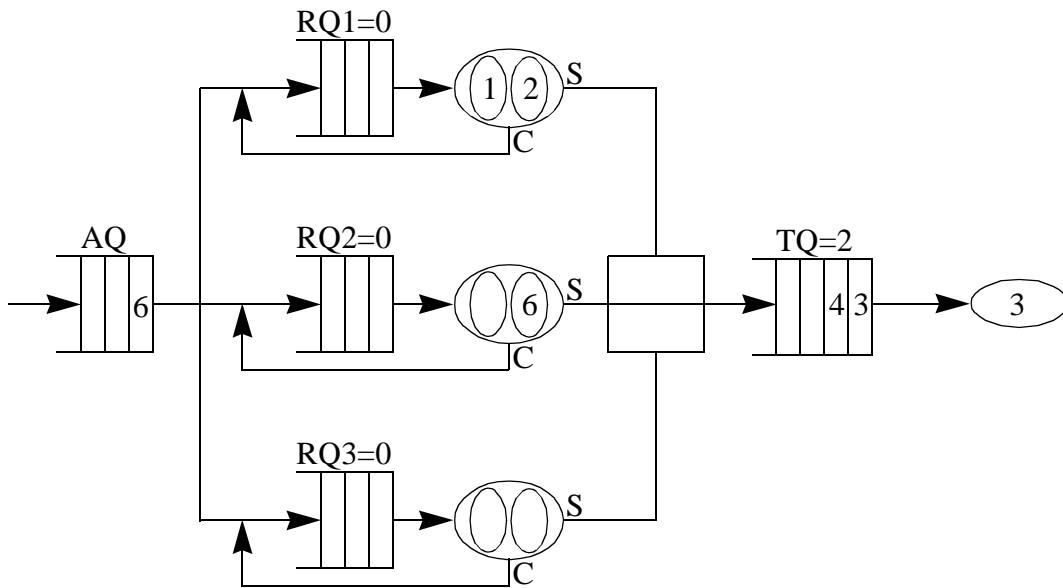


Figure 20 Global TQ Example: RQ2 Active in the Second Cycle

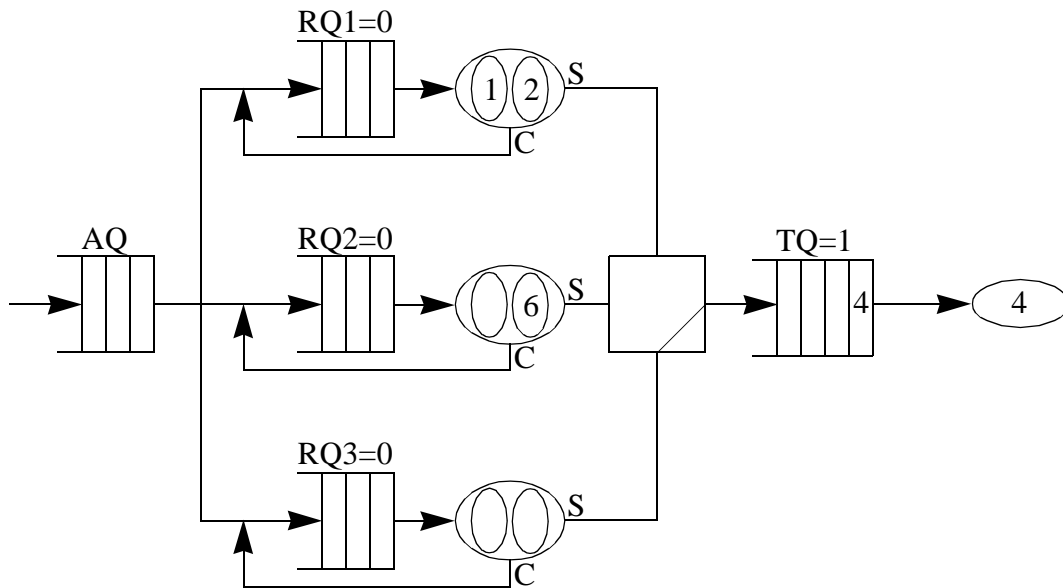


Figure 21 Global TQ Example: RQ3 Active in the Second Cycle

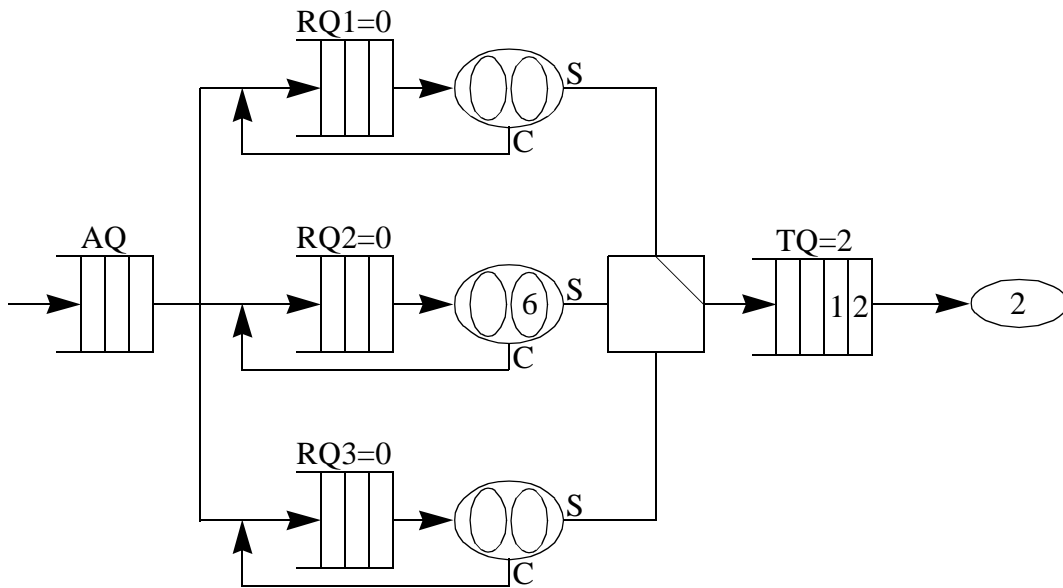


Figure 22 Global TQ Example: RQ1 Active in the Third Cycle

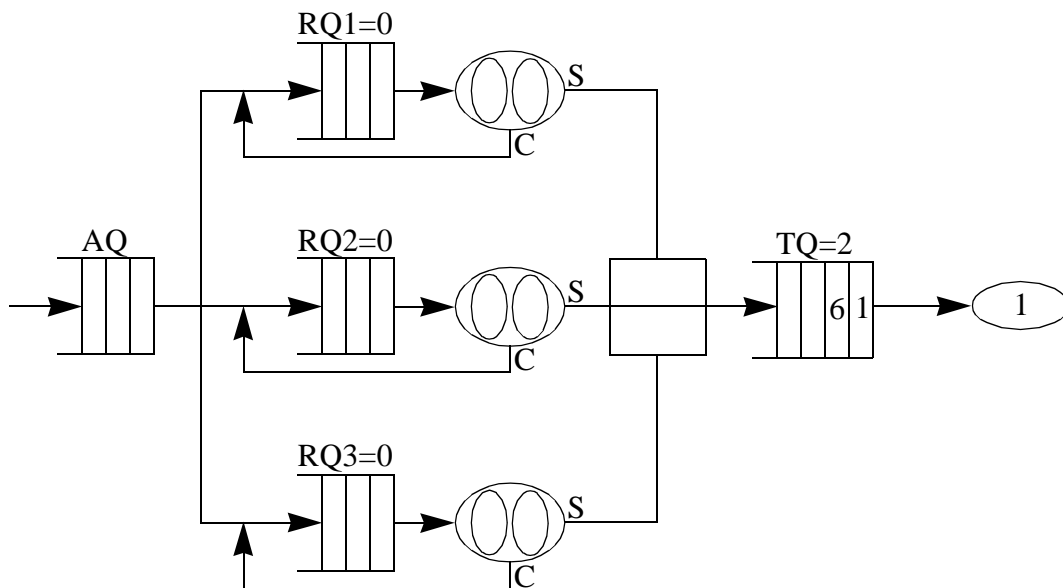


Figure 23 Global TQ Example: RQ2 Active in the Third Cycle

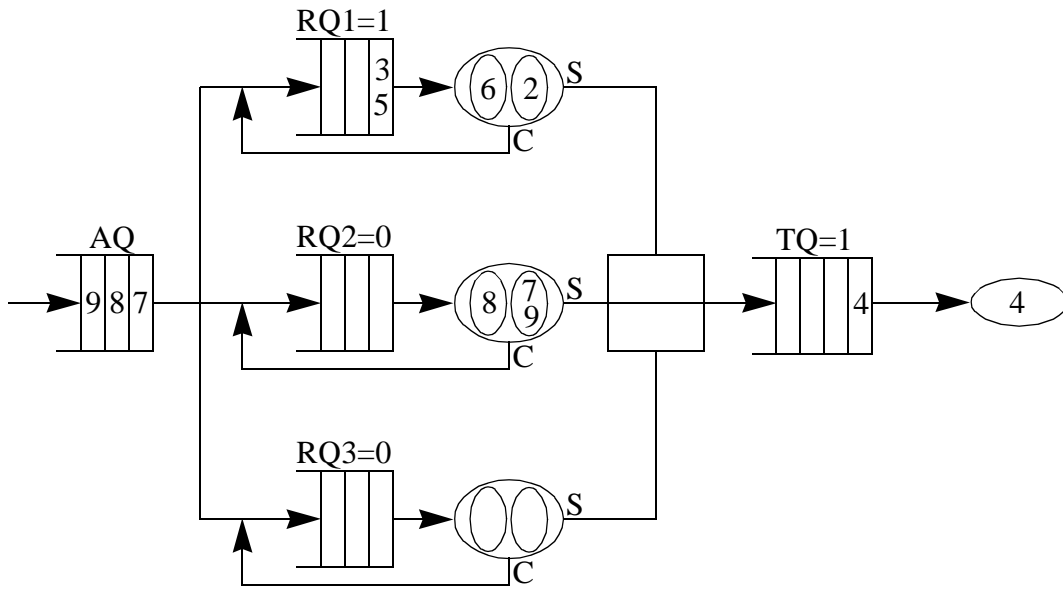


Figure 24 Standard Global TQ QDR

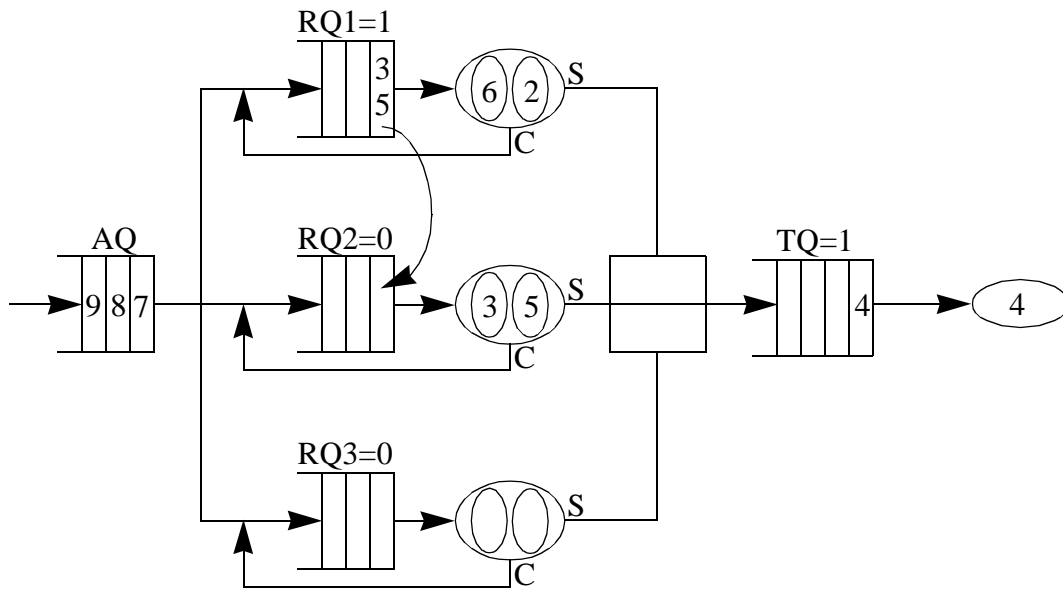


Figure 25 Improved Global TQ QDR

```

Improved_Global_TQ_QDR(int group)
{
    int k, n_request;
    for (n_request = 0, k = 1; k <= m; k++)
        if (Fk == S)
            n_request++;
        else if (Fk == C)
            n_request += 2;
    for (k = 1; k <= m; k++) {
        if (Fk == S) {
            if (Prev_TQgroup == 0 && n_request == 1)
                /* Immediate access */
            else
                TQ++;
        }
        else if (Fk == C)
            RQgroup++;
    }
    Prev_TQgroup = TQ;
    if (RQgroup == 0) { /* This is the improved part */
        for(k = group % interleaving_factor + 1; k != group; k = k % interleaving_factor + 1)
            if (RQk > 0) {
                RQk--;
                RQgroup = 1;
                break;
            }
    }
}

```

## 5 Global TQ in a Dual Bus Environment

A dual bus environment is described in Figure 26. When a node wants to send data, it transmits on the inbound channel, and all nodes receive data on outbound channel. The distances between nodes and the headend are different..

The difference between the channel model in Figure 1 and Figure 26 is that in Figure 1 all of the nodes receive the same slot at the same time, so when they apply QDR to modify TQ and RQ<sub>i</sub>, every node has the same values of TQ and RQ<sub>i</sub> at the same time. But in Figure 26, we see that the nodes closer to the headend receive the same slot earlier, if each node applies QDR to update TQ and RQ<sub>i</sub> immediately, this will result in different values of TQ and RQ<sub>i</sub> at the same time. This will result in a problem when RTR and DTR are used to send requests and data. This problem is solved by using the approach used in DQLAN. Every node uses a procedure to deter-

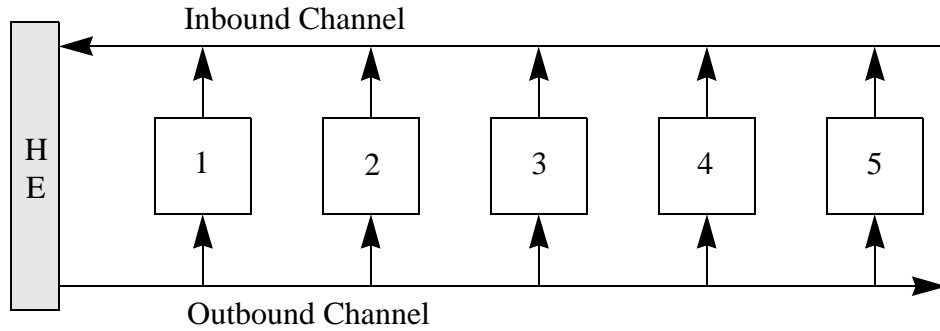


Figure 26 Dual Bus Channel Model

mine the propagation delay between itself and the headend, say  $T_{pi}$ . When each node receives CMS' feedbacks from outbound channel, it registers the feedback information in the buffer and apply the information to update TQ and RQ later. Let a slot time be  $T_s$ . When node  $i$  receives a complete slot, including CMS' and data slot, at time  $t$ , the feedback information can be used to update TQ and RQ at time  $t + (\text{interleaving\_factor} - 1) * T_s - 2T_{pi}$ . This is equivalent to that of a node receiving the slot transmitted by itself after  $(\text{interleaving\_factor} - 1) * T_s$ .

Figure 27 to Figure 34 demonstrate how the protocol operates. This example assumes that the interleaving factor is 5. Stn1 and stn2 have 1 and 2 slot time distance from headend respectively. In stn1, since the feedback received from outbound channel must delay  $(5 - 1) * 1 - 2 * 1 = 2$  slot times, so the number of buffers required to record feedback information is 3. The received feedback information is put into the bottom of the buffer, and is moved up one entry per slot time. The QDR always uses the information in the top of the buffer to update TQ and RQ. The information status in the buffer are E: empty, S: single, and C: collide. In this example, the underlined RQ is the active one. In Figure 27, stn2 sends request and data. In Figure 28, both stn1 and stn2 send requests and data in different slots. In Figure 29, the collided data slot appears in the outbound channel, but stn1 still has not received it yet. Stn1 sends another request and data in the inbound channel. In Figure 30, stn1 receives the first slot it sent and places the feedback status into the bottom of the buffer. In Figure 31, stn1 receives the second slot and places the feedback status into the bottom of the buffer again. We can see the previous status has moved up. At the same time, stn2 receives the first slot, and the feedback status is put into the buffer. In Figure 32, stn2 uses the feedback status in the buffer to apply QDR to update TQ and RQ1. There are two single status' in the CMS thus TQ is incremented by 2. After that, another slot feedback status is put into the buffer. Stn2 is in the second entry of the TQ, so it cannot send data in this slot. In Figure 33, stn1 updates TQ and RQ2 by using the feedback information in its top of the buffer. Since stn1 is in the first entry of the TQ, it sends data in this slot. At the same time, stn2 reduce TQ by 1, and apply QDR to update TQ and RQ2. Since there is a collision in first CMS, RQ2 is incremented by 1. By

using RTR and DTR, stn 2 sends request and data. In Figure 34, stn1 has no data to send, but the second packet it sent out is in RQ3, so it sends a request again.

Following is the modified global TQ protocol for dual bus environments:

```
Dualbus_Global_TQ_Protocol()
{
    int i, j, group, n_buf;
    char Feedback[MAX_BUF][3];
    TQ = 0;
    for (i = 1; i <= interleaving_factor; i++) {
        RQi = 0;
        Prev_TQi = 0;
    }
    n_buf = number of buffer required to record feedback status
    for (i = 0; i < n_buf; i++)
        for (j = 0; j < m; j++)
            Feedback[i][j] = E;
    group = 1;
    while(TRUE) {
        Global_TQ_QDR(group);
        Global_TQ_RTR(group);
        Global_TQ_DTR(group);
        Dualbus_Detect_CMS_Feedback();
        group = (group % interleaving_factor) + 1;
    }
}
```

```
Dualbus_Detect_CMS_Feedback()
{
    int i;
    for(i = 1; i <= m; i++)
        Feedback[n_buf - 1][i - 1] = Fk;
}
```

```
Global_TQ_QDR(int group)
{
    int k, n_request;
    for (n_request = 0, k = 0; k < m; k++)
        if (Feedback[0][k] == S) /* Check info in top of the buf */
            n_request++;
        else if (Feedback[0][k] == C)
```

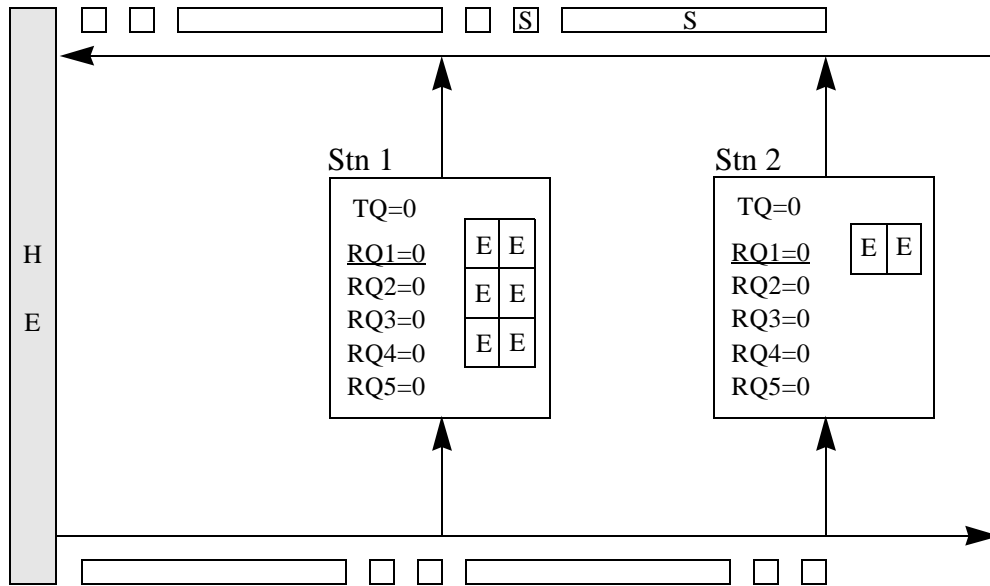


Figure 27 Dual Bus Example: Stn2 Sends Request and Data

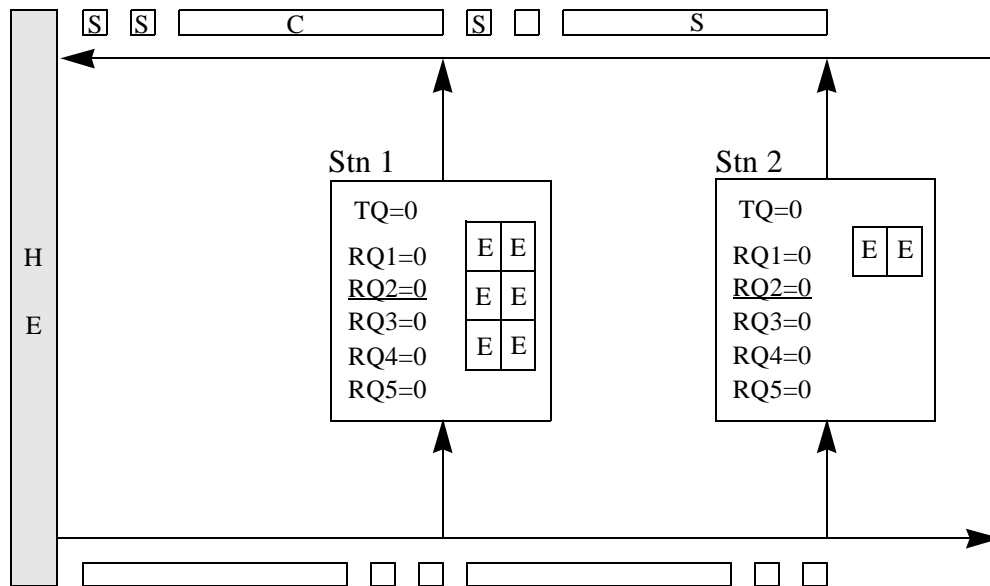


Figure 28 Dual Bus Example: Stn1 and Stn2 Send Request and Data

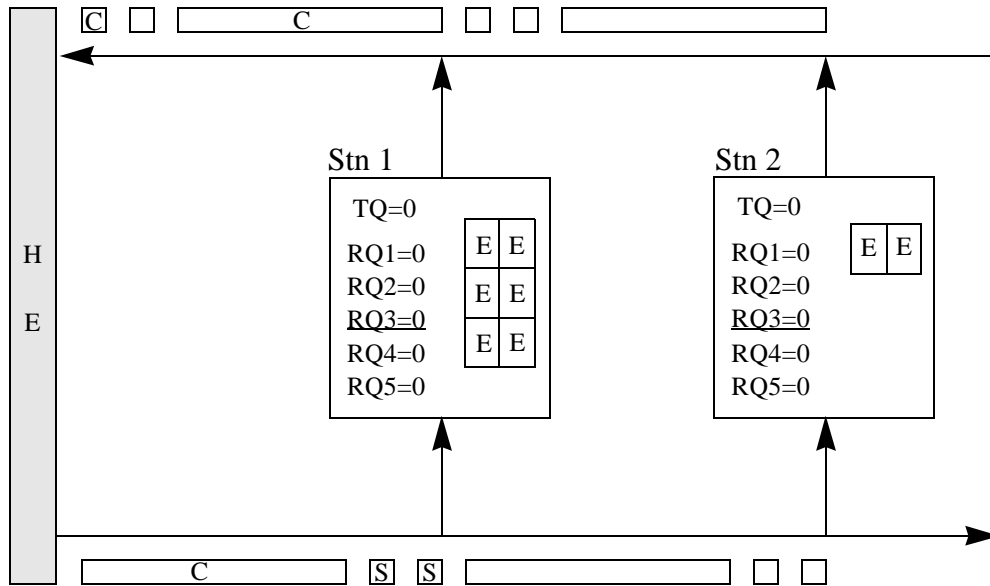


Figure 29 Dual Bus Example: Stn1 Sends Request and Data

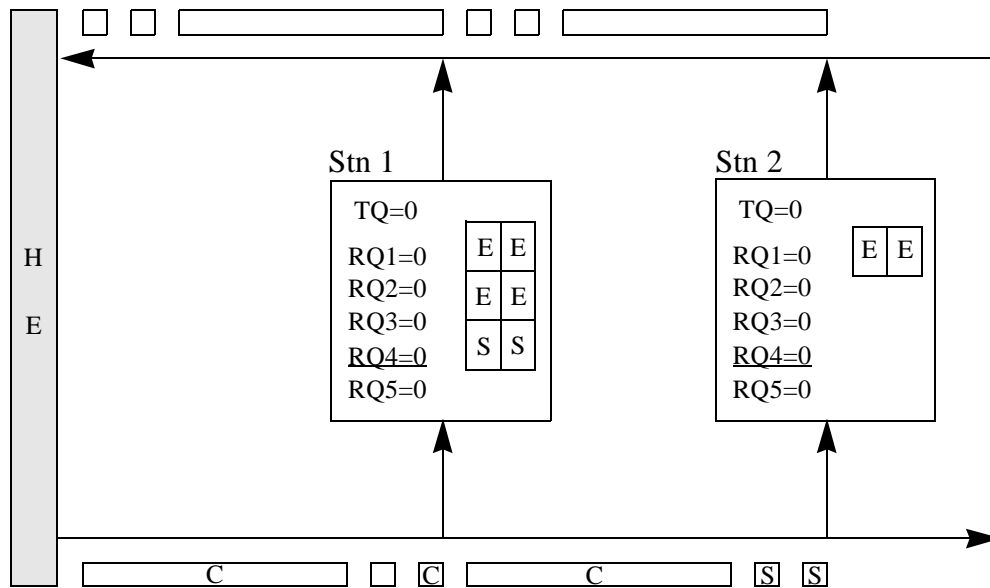


Figure 30 Dual Bus Example: Stn1 Receives Feedback

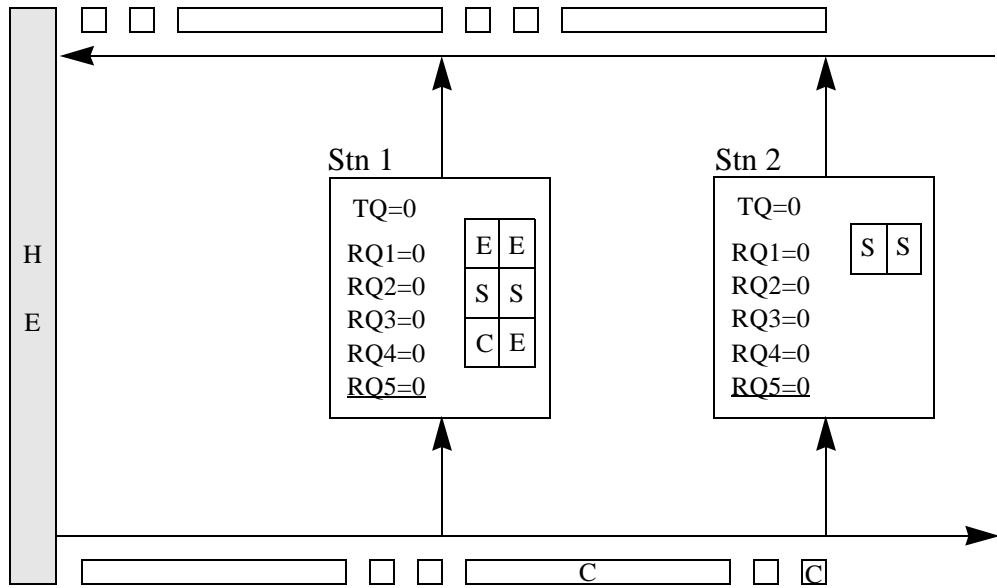


Figure 31 Dual Bus Example: Stn1 and Stn2 Receive Feedback

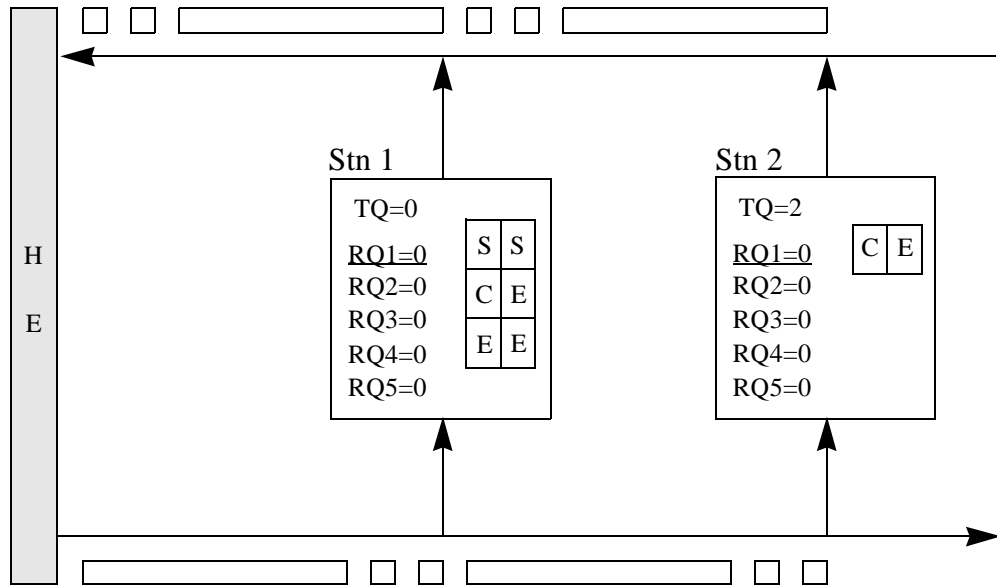


Figure 32 Dual Bus Example: Stn2 Updates TQ

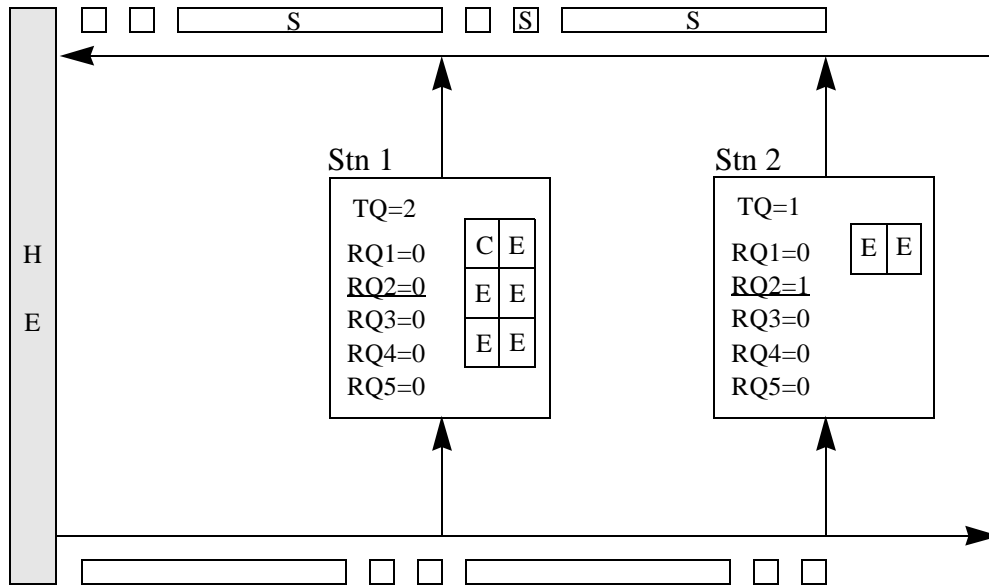


Figure 33 Dual Bus Example: Stn1 Updates TQ and Stn2 Updates RQ2

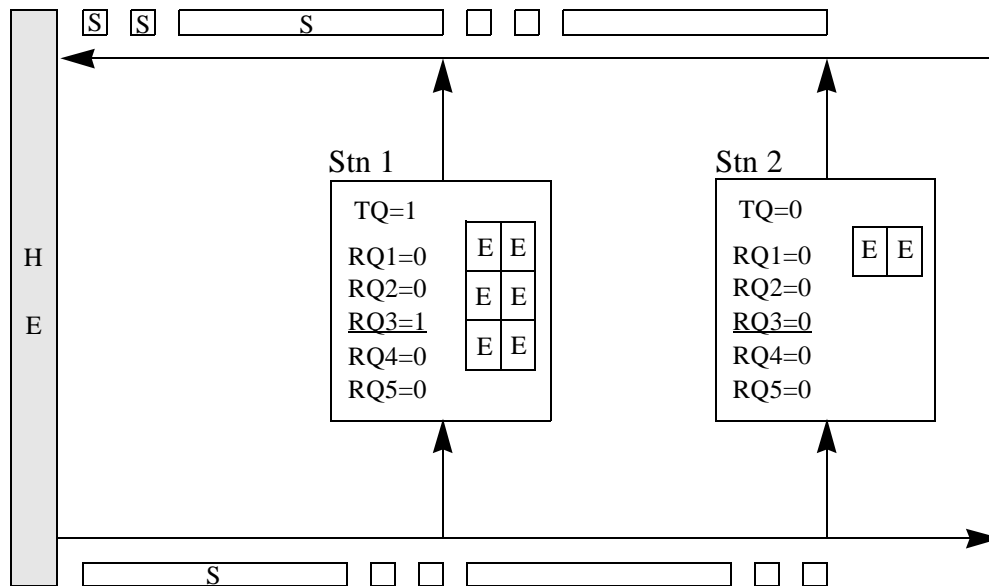


Figure 34 Dual Bus Example: Stn1 Updates RQ3

```

    n_request += 2;
for (k = 0; k < m; k++) {
    if (Feedback[0][k] == S) {
        if (Prev_TQ_group == 0 && n_request == 1)
            /* Immediate access */
        else
            TQ++;
    }
    else if (F_k == C)
        RQ_group++;
}
Prev_TQ_group = TQ;
for(i = 1; i < n_buf; i++) /* Move feedback info upward */
    for(j = 0; j < m; j++)
        Feedback[i - 1][j] = Feedback[i][j];
}

```

## REFERENCES

- [1] J.L. Massey, "Collision Resolution Algorithm and Random Access Communications," Multiuser Communications Systems, G. Longo Ed. New York: Springer-Verlag, 1981. pp 73-137 .
- [2] W. Xu and G. Campbell, "A Distributed Queueing Random Access Protocol for a Broadcast Channel", Computer Communications Review, Vol. 23, No 4, Oct. 1993, pp 270-278. (Conference Proceedings of SIGCOMM '93.)