# [1]在DQDB下一個動態傳送機制
# Dynamic Priority Transmission Mechanism for DQDB

陳俊麟　　　　張瑞雄
Chin-Ling Chen and Ruay-Shiung Chang

*Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan, Republic of China*

*摘要*

*本論文提出一個在DQDB網路上動態優先權的存取協定。 此協定可解決在傳送時，高優先權資料受到低優先權資料干擾的問題。*
*關鍵字：分散式佇列雙巴士, 優先權, 效能。*

*Abstract*

*To improve the ineffectiveness of the priority scheme in DQDB networks, we propose a Dynamic Priority Transmission Mechanism. The mechanism implements preemptive priorities. The higher priority node preempts lower priority node whenever necessary.*

***Keywords*：*DQDB, priority, performance***

## 1. Introduction

The Distributed Queue Dual Bus (DQDB) is the IEEE 802.6 standard [3] for metropolitan area network (MAN). DQDB intends to integrate data, voice and video traffic on a single platform. The DQDB network is based on a pair of unidirectional buses that transmit in opposite direction as shown in Fig. 1. To send in one bus, a node must make request first in another bus. For example, in Fig. 1, if node 2 wants to send to node *N*, then the upper bus will be the data bus and the lower bus will be the reservation bus for making requests. However, the DQDB protocol has several problems in the transmission of prioritized, time-critical data.

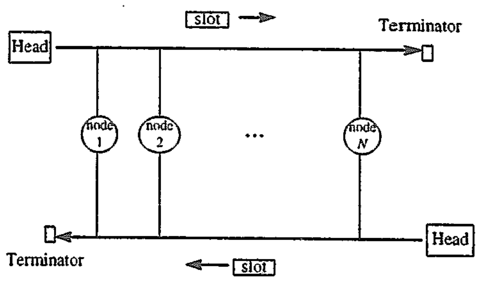

Fig. 1. Basic architecture on DQDB

One shortcoming in DQDB is that the current multiple priority mechanism is not able to provide guaranteed real time performance. This is due to the fact that DQDB protocol does not provide individual station with a global view of priority operations. Each station decides the priority of its own generated message according to the local queue status. Actually, it does not care about and is unable to see the situations of other stations. This leads to improper priority mapping in the global distributed queue. A proper priority assignment scheme is important in DQDB to guarantee the required data be arrived in real time and the delay be bounded.

Another shortcoming in basic DQDB is that the nodes with higher priority cannot satisfy all their bandwidth requirements. If a high priority node is located downstream and a low priority node is located upstream in the reservation bus, the upstream node has more chance to get bandwidth of the network because it can request first. This leads to unfairness problem since in common sense all nodes have to leave as much available bandwidth as possible to the highest priority nodes

[1] This research is supported in part by NSC under contract numbers NSC86-2622-E-007-001 and NSC86-2213-E-011-056.

in the network.

Bisdiskian and Tantany [1] addressed three factors that result in the unfairness of high priority traffic. The first one is the natural ordering of the nodes on the unidirectional buses. The node closer to head of bus has better chance to get the bandwidth. Second, a node must send a request for each segment to be sent. No matter how many segments the node would like to send it has to send them one by one after each successful request. The last one is that traffic from all priority levels (but from different nodes) can be allowed to access the network at the same time. Instead of allowing all priorities to present in the network simultaneously, they proposed a new method to allow the higher priority node to use the network as long as it needs. The new idea is called priority hold. However, the paper did not specify the way for the low priority node to renew its request after its low priority traffic is preempted by the high priority traffic.

Kamal and Bissonauth [5] proposed another priority mechanism by introducing the concept of a virtually empty slot. Virtually empty slots are defined as those slots that are either empty or occupied by a lower priority segment. This mechanism enables higher priority nodes with heavy load to use virtually empty slots and shut out lower priority nodes completely. However, to implement it, three queues for each priority in one node are needed. Each node must be equipped with two additional counters, and a pointer to each queue is required. Besides this data structure overhead, the nodes located downstream still suffer a penalty because of their location. The priority mechanism is a minor modification to the DQDB protocol and its finite state machine is similar to that of the standard DQDB.

Huang and Wu [3] used priority promotion scheme to solve the unfairness problem in DQDB network. To prevent the downstream nodes in the reservation bus from starvation in another bus, the downstream nodes promote their priority when the number of passing empty slots exceeds some certain value. When a node has promoted its priority to the highest, it is allowed to send out its buffered packet on the first empty slot. Notwithstanding, there are drawbacks in this scheme. First, it did not consider about the basic DQDB priority mechanism. The format of Access Control Field (ACF) and the different priority levels of request counters and countdown counters were not mentioned. Second, the authors did not describe the detailed operations of segment transmission in each node.

An important problem in supporting high priority service over computer network is Quality of Service (QoS) management. QoS management strategy refers to allocation of network resources so as to guarantee good performance for time-critical data. To improve the ineffectiveness of the priority scheme in DQDB networks, we propose a mechanism to implement preemptive priorities. This mechanism provides efficient QoS guarantee for high priority service. The high priority service will be served immediately. The low priority service uses the remaining bandwidth as is available. To improve the shortcomings of previous papers mentioned above, combined techniques such as priority preemption and priority promotion are used to achieve better performance. Moreover, the detailed design of node architecture is also provided.

The rest of the paper is organized as follows. In Section 2, the basic DQDB protocol is described. Cell format and the key ideas behind the proposed mechanism are explained in Section 3. The analysis and simulation results of comparison between the existing priority mechanisms and the new priority mechanism are shown in Section 4. Finally, Section 5 concludes this paper.

## 2. Basic DQDB protocol for priority mechanism

A DQDB network of size $N$ consists of $N$ nodes. These nodes, numbered 0 to $N$-1, are evenly distributed over the network. The head of bus (HOB) generates empty slots periodically. Each slot has 53 bytes and consists of a segment payload and one byte of access control field (ACF) (Fig. 2). Each node has three sets of *req_ctr* (request counter) and *cd_ctr* (countdown counter) for three priority levels respectively. Operation of *req_ctr* and *cd_ctr* [4] involves read operations on Busy bit and SL_TYPE (slot type) bit in ACF on data bus and read operations on the Request field in ACF on reservation bus. A request for access to data bus at priority level $i$ is signaled to other nodes by a write operation into the *req_i* bit in the ACF in one of the slots passing on the reservation bus. The write operation succeeds and stops after it set to one the first zero *req_i* bit on the reservation bus.

DQDB Standard

| | | | | |
|---|---|---|---|---|
| Busy (1) | SL_TYPE (1) | PSR (1) | Reserved (2) | Request (3) |

Proposed

| | | | | |
|---|---|---|---|---|
| Busy (1) | SL_TYPE (1) | preempt (1) | Priority (2) | Request (3) |

Fig. 2 ACF format

Each node with outstanding segments to send requests one at a time by writing *req_i* in ACF on request bus. When a slot arrives in request bus, a node read the non-zero *req_i* in the Request field of ACF and compares it with its own priority $j$. *req_ctr*($j$) increments by one for any *req_i* bit on the reservation bus whose priority $i$ is greater than or equal to $j$. *req_ctr*($j$) decreases by one for an empty slot passing on data bus.

When a node is ready to send, it enters Countdown State. The node copies the value of *req_ctr(j)* to *cd_ctr(j)* and reset *req_ctr(j)* to zero. *cd_ctr(j)* increments by one on seeing any *req_i* bit on the reservation bus whose priority *i* is greater than *j*. *cd_ctr(j)* decrements by one for an empty slot passing on data bus. The node has the right to access the empty slot when *cd_ctr(j)* reaches zero.

## 3. New priority mechanism

To improve the shortcoming of [3] mentioned in Section 1, the ACF format and detailed node architecture of new mechanism are proposed in Subsection 3.1 and Subsection 3.2. Subsection 3.3 proposes a new priority mechanism to ameliorate the weakness of non-retransmission for the preempted low priority node in [1].

### 3.1 ACF Format

The ACF in basic DQDB protocol currently contains three request bits and two bits reserved for future use. These two reserved bits are used in the proposed scheme and are called priority field (PF) (Fig. 2). 00 represents the lowest priority. 10 or 11 is the highest priority and 01 is the second highest priority. Three request bits correspond to three priority levels. In the basic DQDB protocol, the Previous Slot Read (PSR) bit indicates whether the previous slot has been read or not. When a segment arrives at the destination, the destination node sets PSR bit in the following slot. An erasure node buffers an entire slot and the following slot for reading the PSR bit. If the PSR bit is set, the previous slot is erased. The downstream nodes, therefore, may reuse this empty slot. In our proposed scheme, each node may preempt a slot if necessary. The erasure node is not needed.. PSR bit, therefore, is modified to be Preempt bit. Preempt bit is used to inform the node status, preempt (1) or idle (0), to the nodes located upstream in the data bus. When a higher priority node positioned downstream executes preemption operation, it sends Preempt bit (1) and priority field upstream by the reservation bus. Whenever a lower priority node positioned upstream in the data bus sees the Preempt bit (1) and higher priority bits, it increases a counter, called *preempt_ctr*, by one. On seeing an empty slot in the data bus, the node decreases *preempt_ctr* by one. After *preempt_ctr* reaches zero, the node follows the basic DQDB protocol to get the right to transmit.

### 3.2 Node Architecture

Each node is assigned a fixed priority level *k*, where *k* = 0, 1 or 2. There are two processing functions in each node: reception processing and transmission processing (Fig. 3). Reception processing is in charge of the handling of arriving slots. This function first delineates the arriving slot type (data and/or request) by reading out the ACF format. It passes request slot to transmission processing function and executes preemption to alter data when necessary. By the preemption operation, the higher priority node has more chance to access the empty slots and avoid the penalty due to positioning [1]. This preemption condition will be described below. A request counter (*req_ctr*) and a countdown counter (*cd_ctr*) are in charge of the operation for accessing empty slot in the node. Only one set of request counter (*req_ctr*) and countdown counter (*cd_ctr*), rather than three sets of counters [5], is needed in each node. Preempt counter is used to count the number of slots which are preempted downstream in the data bus. Transmission processing updates request counter and countdown counter when receiving higher or equivalent priority request slots. It also deals with the injection of slot into bus. In Fig. 3, there are two types of queues in each node: input queues and an output queue. Input queues hold its own outstanding segments and the arriving segments from other lower priority nodes that have not reached their destination. Input queues are composed of *k*+1 priority queues for storing *k*+1 priority levels outstanding segments for a node with priority *k*, where $k \neq 0$. Since priority 0 nodes do not store any segments from other nodes, the *queue*(0) in the priority 0 node is not required. Denote the *i*th priority queue as *queue*(*i*) for $0 \leq i \leq 2$.

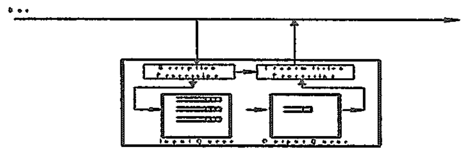

Fig. 3 Node access control block

For instance, a node with priority 2 has three priority queues in input queues. The *queue*(2) stores the outstanding segments for itself while *queue*(1) and *queue*(0) hold the lower priority segments from others. Each priority queue has a queue length counter, *queue_ctr*(*i*). Two successive priority queues ((*queue*(*i*) and *queue*(*i*-1)) are connected by a switch, which is controlled by the queue length counter *queue_ctr*(*i*-1). *queue_ctr*(*i*-1) records how many segments are in the (*i*-1)th queue. Whenever *queue_ctr*(*i*-1) value reaches a maximum value, the switch between *queue*(*i*-1) and *queue*(*i*) is connected to *queue*(*i*) (Fig. 4). Then, all the segments in *queue*(*i*-1) are moved forward to *queue*(*i*) until *queue_ctr*(*i*) reaches its maximum value or *queue_ctr*(*i*-1) is equal to zero. After the operation of movement is completed, the link is disconnected and the switch goes back to the original position. The characteristics of maximum length value of a queue will be discussed in Property 1. Depending on the positions of the nodes, each node may be configured with a different value.

Fig. 4 is an example for a priority 2 node. Whenever *queue_ctr*(0) reaches its maximum value, the switch of *queue*(0) is connected to *queue*(1). All the segments in *queue*(0), therefore, are moved to *queue*(1) till the *queue_ctr*(1) reaches its maximum value or *queue*(0) is empty. The same procedure is applied to the switch between *queue*(1) and *queue*(2). When there are any segments in *queue*(2), those segments will be moved forward to output queue one by one. Output queue holds only one segment for moving to the transmission processing one at a time.

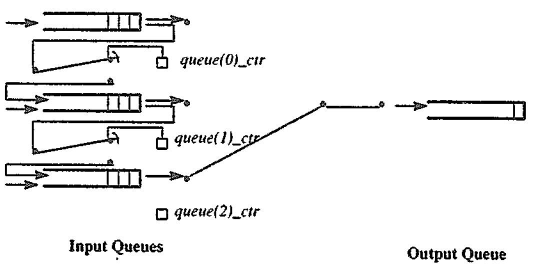

Fig. 4 Example of input queue and output queue

***Property 1***: Let *D(p,q)* be the maximum distance from a priority *p* node to priority *q* node, where priority *p* node is positioned upstream with respect to priority *q* node in the data bus and *p*<*q*. Distance between nodes is estimated in slot time. Then the maximum possible length of queue(*p*) in node *q* is 2**D(p,q)*.

*Proof:* After *D(p,q)* time, the first data slot of *p* arrives at *q*, which is then preempted and stored in queue(*p*) and a preempt signal is sent. After *D(p,q)* time, this preempt signal arrives at *p*, which then stops sending. Therefore, the number of slots sent by *p* is 2**D(p,q)*, which must be queued at *q*. Q.E.D.

## 3.3 Priority Implementation

### 3.3.1 Reception Processing Function in Data Bus

Suppose a busy slot that contains a segment of priority *m* arrives at a node with priority *k*, which is not the destination in the data bus. Comparison of *m* and *k* in the reception processing function, there are two cases:

Case (1): *k*>*m*. The node executes preemption mechanism. The node first copies the segment of priority *m* into *queue(m)* and then *queue_ctr*(*m*) is increased by one. The slot is then emptied. Suppose the maximum length for each priority queue is *L*. Whenever *queue_ctr*(*m*) reaches *L*, it promotes its priority by moving segments from *queue*(*m*) to *queue*(*m+1*) until *queue_ctr*(*m*+*1*) is equal to *L* (Fig. 4).

Case (2): $k \leq m$. The node with priority *k* realizes that there is higher priority *m* traffic in the network. The node reacts by ceasing its own transmission procedures. Reception processing function lets the busy slot pass to transmission processing function. The slot then goes forward to the bus.

### 3.3.2 Transmission Processing Function in Data Bus

Now consider the transmission processing function in the node of priority *k* for the data bus. When an empty slot arrives, the transmission processing function first checks the value of *preempt_ctr*. If the *preempt_ctr* is greater than zero, the node lets empty slot pass and *preempt_ctr* is decreased by one. When *preempt_ctr* reaches zero, the node follows the basic DQDB protocol to get the right to transmit. The segment in *queue*(*k*) is moved forward to output queue one at a time. After that, the segment in the output queue is injected into the bus by the transmission processing and heads for its destination. Whenever one segment from *queue*(*k*) enters the output queue, *queue_ctr*(*k*) is decreased by one. The switch changes its connection to *queue*(*k*-1) when *queue_ctr*(*k*) reaches zero. If a new priority *k* segment arrives, the connection will be switched back to *queue(k)*(Fig. 5).

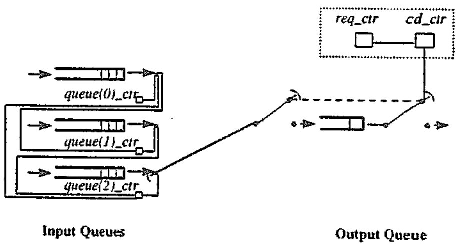

Fig. 5 Switching mechanism for input queues and sending mechanism for output queue

### 3.3.3 Reception Processing Function in Reservation Bus

Suppose a busy slot arrives at the node (priority *k*) in the reservation bus. The reception processing function first determines whether the slot contains a Request bit of priority *m* (*req_m*) or a Preempt bit. If the slot contains Preempt bit and with higher priority bits, *preempt_ctr* is increased by one. This node knows that a higher priority node positioned downstream in the data bus wants to transmit. The node stops transmission. If the slot contains a Request bit (*req_m*), reception processing compares $m$ and $k$ and there are two cases:

Case (1): $k > m$. The node ignores the lower priority request bit according to priority mechanism. It does not update the request counter and countdown counter.

Case (2): $k \leq m$. Basically, this case follows standard DQDB protocol. There are only one set of *req_ctr* and *cd_ctr* in the node. In the idle state, the node increments *req_ctr* by one on seeing an incoming slot with *req_m*. While *req_ctr* is greater than zero and one empty slot arrives on data bus, the value of *req_ctr* is decreased by one. When a node begins the countdown, the value of the *req_ctr* is copied to *cd_ctr* and the value of *req_ctr* is reset to zero. *cd_ctr* is decreased by one on seeing each passing empty slot. When *cd_ctr* is zero, the node gets the right to send a segment.

### 3.3.4 Transmission Processing Function in Reservation Bus

Consider the transmission processing function in the node of priority $k$ for the reservation bus. The transmission processing function checks whether the reception processing function executes preemptive operation. As long as this node executes preemption, it sets preempt bit to 1 and sets priority bits. The slot carrying preempt bit is sent upstream by the reservation bus. Otherwise, the transmission processing function set the corresponding request to 1 when the node wants to transmit. The slot is then injected to the reservation bus and passed upstream.

## 4. Simulation and Performance Evaluation

In this section, we scrutinize an analytical model and study the delay as well as throughput characteristics of the proposed protocol by simulation. For each case considered, we also provide the simulation results according to methods in standard DQDB and Kamal and Bissonauth's [5] for comparison.

The simulation result of Fig. 6 illustrates the network throughput property. We evaluate these three schemes' throughput based on the different message length. The proposed one has larger average throughput than the others do. When the message length increases, the average throughput of the proposed scheme is still better than those of the other two.

The average message delay for each node under different network load is shown from Fig. 7-1 to Fig. 7-3 for these three schemes, respectively. The message delay of low priority traffic in the proposed scheme is close to those of the other two in the light load. On the other hand, the message delay of high priority traffic does not increase as the network load increases. However, the message delay of lower priority traffic increases dramatically as the network load increases.

In Fig. 8, the average throughput for the three schemes have no difference in light load. However, the proposed scheme outperforms the other two as the workload increases.

To conclude from the simulation results, the proposed scheme and [5] permits the highest priority class to be better serviced under heavy load. Lower priority classes get to transmit only when the higher priority class is inactive. However, the performance of [5] shows longer delay and lower throughput when compared with that of the proposed scheme under heavy load. The complicated mechanism and three sets of queue for each priority in one node are the major reason.

## 5. Conclusions

We presented a new Dynamic Priority Transmission Mechanism for DQDB. By modifying the nodal structure and hardware of basic DQDB, the system performance is improved compared to other schemes. We have the preempt field in place of PSR field and use two reserved bits for the priority bits. We also study the delay and throughput characteristics via simulation, compared to the standard IEEE 802.6 DQDB and [5]. The new mechanism provides higher throughput efficiency and lower delay required for the higher priority class traffic.

It is widely acknowledged that the support of multi priority level is required to provide a variable quality of service to the network. Support of high-priority traffic is also needed for network control and management. The new generation of computer applications, such as multimedia conferencing and remote video, have widely varying QoS requirements. ATM, another integrated service network, are designed to support various applications. In the case of interworking between these two kinds of networks, a QoS mapping must be defined to meet the requirements of all the application. The defining of QoS metric between networks is complicated. However, it is an important issue in the future.

## References

[1] C. Bisdiskian and A. N. Tantany, "A Mechanism for Implementing Preemptive Priorities in DQDB Subnetworks," *IEEE Transactions on Communications* Vol. 42, No. 2/3/4 1994, pp. 834-839

[2] D. Gross and C. M. Harris, *Fundamentals of Queuing Theory*, New York: John Wiley and

Sons, Inc. 1994.

[3] T. Y. Huang and J. L. C. Wu, "Priority Promotion DQDB networks to improve fairness," *Computer Communication* Vol. 17, No. 5, May 1994, pp. 332-338.

[4] IEEE Proposed Standard 802.6, *Distributed Queue Dual Bus (DQDB)*, Subnetwork of a Metropolitan Area Network (MAN), Oct. 1990.

[5] A. E. Kamal and A. Bissonauth, "Priority Mechanism for the DQDB network," *IEE Proc.-Communi.* Vol. 1141, No. 2, April 1994, pp. 98-104.

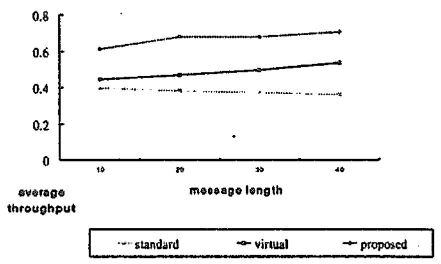

Fig. 6 Average throughput for 5 nodes network, 150 Mbps, Poisson arrival, internode distance = 1 slot-time (2.726 $\mu s$), $\rho$=1.0

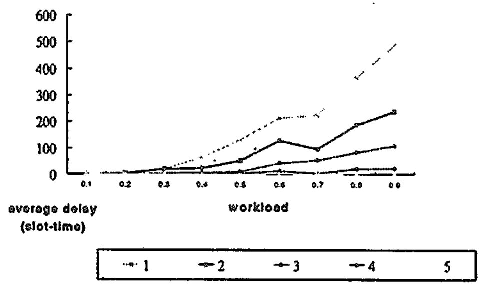

Fig. 7-1 Average delay in DQDB for 5 nodes network, 150 Mbps, Poisson arrival, message length = 40 (slot), internode distance = 1 slot-time (2.726 $\mu s$)

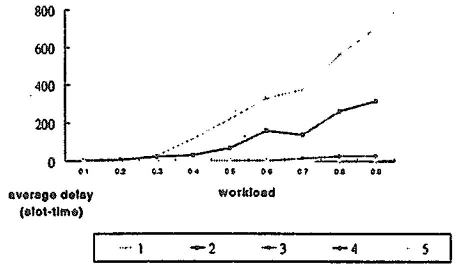

Fig. 7-2 Average delay in Virtual Slot Scheme for 5 nodes network, 150 Mbps, Poisson arrival, message length = 40 (slot), internode distance = 1 slot-time (2.726 $\mu s$)

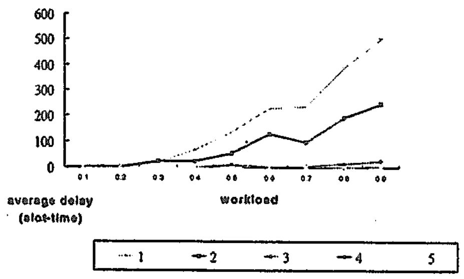

Fig. 7-3 Average delay in Proposed Scheme for 5 nodes network, 150 Mbps, Poisson arrival, message length = 40 (slot), internode distance = 1 slot-time (2.726 $\mu s$)

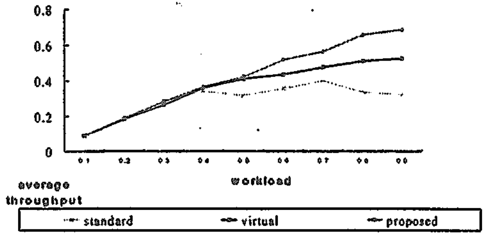

Fig. 8 Average throughput for 5 nodes network, 150 Mbps, Poisson arrival, message length = 40 (slot), internode distance = 1 slot-time (2.726 $\mu s$)