REAL-TIME COMMUNICATION IN LOCAL AREA RING NETWORKS

Q. Zheng and K. G. Shin
Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122
E-mail: {zheng,kgshin}@eecs.umich.edu

Abstract

Timed token protocols are almost exclusively used for medium access control in local area ring networks. We consider in this paper the feasibility of using buffered transmission which has the advantages of eliminating the token passing overhead and relieving the network interface from the complex token handling operations. By establishing real-time channels (each of which is a unidirectional virtual connection), the end–to–end delivery delay of real-time packets can also be controlled more flexibly than the timed token protocols.

We first review the concept of real-time channel (RTC) and the RTC establishment procedures. Its performance is then compared with the FDDI timed token protocol. Our simulation results show that the buffered transmission in local area ring networks equipped with real-time channels enhances both the networks' throughput and the ability to support heterogeneous real-time traffic. The implementation of the network interface is also examined with an example design showing the feasibility of real-time channels in high-speed local area networks.

1 Introduction

The ring topology has been widely used in Local Area Networks (LANs). Two well-known examples of these are the IBM token ring and the Fiber Distributed Data Interface (FDDI). Compared to the bus topology like the Ethernet, the ring topology is better suited for high-performance networks operating at a very high transmission rate ranging from 20 to 500 mega bits per second (Mbps) [1]. Unlike the bus topology, a ring network can connect many stations over a long distance without significant performance degradation.

With a proper Medium Access Control (MAC) protocol, a ring network can provide bounded access delays for real-time packets. One such example is the FDDI's timed token protocol which guarantees the maximum access delay for synchronous packets not to exceed two times the preset Target Token Rotation Time (TTRT). However, the bounded access delay provided by a timed token ring is not small enough for a true multimedia LAN [2] or for distributed real-time systems for the following two reasons.

First, the ring latency (defined as the token rotation time in the absence of network traffic) has a significant effect on the maximum network throughput when a stringent access delay bound is required. It was suggested in [3] that to maintain throughput levels of the FDDI above 80 Mbps, the TTRT should be at least five times the ring latency. For the maximum configuration of the FDDI with 1000 stations and a 200 km fiber, the ring latency is about 1.62ms [3]. This requires TTRT $\geq$ 8.1ms. The minimum access delay bound for synchronous packets is then at least 16.2ms, making the network inadequate for those real-time applications with deadlines tighter than this.

The second problem of a timed token ring is its inflexibility in supporting heterogeneous real-time traffic with different delay and bandwidth requirements. This is because the access delay is controlled by the TTRT which is *identical* for all stations. So, all real-time traffic is guaranteed to have the same access delay, resulting in the inefficient use of network band-

*The work reported in this paper was supported in part by the Office of Naval Research under Grants No. N00014–92–J–1080 and N00014–91–J–1115, and the National Science Foundation under Grant No. MIP–9012549. Any opinions, findings, and recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the ' funding agencies.

0742-1303/92 $3.00 © 1992 IEEE

width when some real-time traffic needs stringent delay bounds while others do not.

To alleviate the first problem, Shin and Zheng [4] proposed an improved version of the timed token protocol which can maintain the network throughput to a maximum possible level while guaranteeing the requested access delay bound for synchronous packets. The main idea in [4] was to modify the FDDI's timed token protocol such that the average token rotation time equals the maximum token rotation time when there is a heavy network throughput demand (note that the average token rotation time is *one half* of the maximum token rotation time for a standard FDDI). This is equivalent to reducing the network latency by a factor of two. For the maximum FDDI configuration described above, the modified protocol can provide the access delay bound as small as $16.2/2 = 8.1$ms.

Another way of enhancing the FDDI, called FDDI-II, was proposed to make FDDI suitable for multimedia applications [1, 2]. Realizing the continuous-stream property of the voice traffic, FDDI-II adds a circuit-switched service to the ring, which sets up connections at data rates of multiples of 8 Kbits per second (Kbps). Circuit switching is suitable for constant bit rate traffic, but may waste link bandwidth or cause performace deterioration in case of variable bit rate traffic, e.g, the compressed video transmission.

The fundamental idea behind any token-controlled medium access protocol is to view the whole ring as a single resource, and accesses to the resource are scheduled in a pseudo-centralized manner, i.e., using a single token. Due to the geographical distribution of the stations, any centralized scheduling mechanism will inevitably introduce undesirable overhead. In this paper, we address the feasibility and advantages of using buffered transmission in ring networks. When buffered transmission is used, each link is viewed as an independent resource and link-access scheduling is done at each station in a decentralized manner. Thus, no MAC overhead will occur and the network bandwidth could be used more efficiently than the usual single token approach.

Buffered transmission has long been used in message-passing multicomputers [5, 6] whose point–to–point interconnection networks make any token protocols very inefficient. Transmission techniques like virtual cut-through and wormhole routing together with hardware-implemented switching functions have made the speed of packet transmission fast enough to handle tightly-coupled distributed computations.

One problem with buffered transmission, however, is its inability to support real-time communication. Due to the random queueing delays at each link and multihops a packet must traverse, end–to–end packet delivery delays are difficult to control. Unlike the FDDI, a bare buffered network cannot guarantee the maximum packet delivery delay bound.

To counter the above problem, Ferrari and Verma [7] proposed the concept of *real-time channel* to support time-constrained communication in buffered wide area networks. Real-time channels use two techniques to guarantee end–to–end packet delivery delay bound: admission control of the channels and deadline scheduling of packet transmissions.

Like circuit switching, admission control requires those processes requesting real-time communication to establish real-time channels before starting packet transmission. A channel establishment request may be accepted or rejected, depending on the current network-load condition. Admission control is necessary because packet delay bounds cannot be guaranteed without controlling the network traffic load. Interested readers are referred to [7, 8, 9, 10] for a complete discussion on the conditions and procedures of real-time channel establishment.

Packet transmissions are scheduled as follows. Real-time packets have a higher transmission priority than that of the non real-time packets. Each real-time packet is assigned a deadline over each link it traverses. When several real-time packets contend for use of the same link, the packet with the earliest deadline is transmitted first. There are two advantages of using deadline scheduling as follows.

1. Minimal effects of contention delays: For real-time applications, we need to guarantee the *maximum* packet delay bound. Unlike the average delay, the contention delays at the transmission links have significant effects on the maximum packet delays, even when the traffic load is not heavy. The deadline scheduling policy can minimize the effects of contention delays in the sense that given any set of packets with deadlines, if they are schedulable under any scheduling policy (i.e., every packet can be transmitted before its deadline), so can they under the deadline scheduling policy [11]. Thus, the deadline scheduling policy gives the communication network the maximum capacity to accommodate real-time channels. In other words, deadline scheduling minimizes the probability of rejecting channel establishment requests.

2. Channel protection: Under the deadline schedul-

ing policy, real-time channels are protected from one another. When establishing a new real-time channel, the worst-case queueing delay is calculated for each link under the assumption that all existing real-time channels over this link will generate packets according to their prespecified patterns (i.e., do not exceed their *a priori* specified maximum packet generation rate and maximum packet length). In practice, however, it is possible that some channels exceed their prespecified rates. By properly assigning deadlines to the packets of different channels [8], the deadline scheduling policy can ensure that those channels exceeding their pre-specified rates will not affect the timely delivery of the other channels' packets. Other scheduling policies, like FIFO or priority scheduling, do not possess this property.

In summary, a real–time channel has the performance of a dedicated circuit without monopolizing the circuit. It guarantees the end–to–end delay bound as long as the source node does not generate packets at a rate higher than the pre-specified value. However, unlike a dedicated circuit, a real–time channel does not reserve any transmission bandwidth. The links are free to transmit other non real–time packets whenever there are no packets of the real–time channels to be transmitted. This makes real–time channels efficient for time–constrained communication.

We will in this paper address the performance and implementation issues of real-time channels in buffered local area ring networks. The performance of real-time channels is compared with that of the FDDI token ring in Section 2, showing that real-time channels are better suited to support heterogeneous real-time communication in addition to their improved network throughput. Section 3 discusses the issues in implementing the ring interface to support real-time channels. The paper concludes with Section 4.

2 Performance of Real-time Channels

We first describe briefly a centralized version of real-time channel establishment procedure [9, 8] which will be used to evaluate the performance of real–time channels against that of FDDI timed token ring.

To set up a real-time channel, the requesting process must determine two parameters, T and C, specifying its traffic generation pattern, where T is the minimum packet inter-generation time and C is the maximum packet transmission time. It is reasonable to assume prior knowledge of these parameters for many applications, such as interactive voice transmission and real-time control/monitoring. In other applications where the traffic pattern is less predictable, the estimated values of T and C could be used. A process may exceed its prespecified maximum packet generation rate at the risk that these packets may be delivered with delays longer than the prespecified bound or may even be discarded, but due to the deadline scheduling of packet transmissions, this particular process will not affect the guarantees of the other existing channels.

The process then sends a channel request message containing T and C together with the end–to–end packet delay bound D and addresses of the source and destination nodes to a special node containing the *Network Manager* (NM), which maintains the information of all existing channels and executes the following algorithm to check if the requested channel can be established under the current network-load condition.

Channel Establishment Algorithm:

Step 1. Suppose a new channel is to run over the route from the source to destination that contains k links $\ell_1, \cdots, \ell_k$. Calculate the minimum packet delay bounds d^j_{min} over link ℓ_j, $j = 1, \cdots, k$ under the deadline scheduling policy.

Step 2. If $\sum_{j=1}^{k} d^j_{min} \leq D$, the requested real-time channel can be established. Assign the delay of ℓ_j to be $d^j = d^j_{min} + \delta/k$, where $\delta = D - \sum_{j=1}^{k} d^j_{min}$. Otherwise, the channel establishment request is rejected.

The algorithm for calculating the minimum packet delay d^j_{min} in Step 1 is presented in the Appendix. Interested readers are referred to [9] for a detailed account.

The Network Manager then sends a reply message to the requesting process notifying the acceptance or rejection of the channel request. If the channel request is accepted, the requesting process also gets the calculated delay bounds d^j's over the links with which a packet's deadlines for the links on the route are calculated and added to the header of the packet. Each intermediate node strips off the corresponding deadline field in the header of a packet and uses this information to schedule the transmission of the packet in the presence of other packets. When a real-time channel is no longer needed, it is removed by having the requesting process send a channel removal message to the Network Manager.

We now compare the performance of real–time

channels with that of FDDI token ring in terms of their ability to support both real-time and non real-time communication. The example we are going to use is a typical FDDI dual-ring network with the link transmission bandwidth 100 Mbps. The network has 80 stations with the ring latency of $0.4ms$. Assuming a propagation delay of $5.058\mu s/km$ and a latency of $0.6\mu s$ per station [3], this corresponds to a ring length of 70 km.

The network is designed to support the following three types of traffic:

Non real-time traffic: This type of traffic is generated randomly and does not have any explicit packet delivery delay requirements. Usually, this type represents the main portion of network traffic. The network must be able to support this type of traffic as much as possible.

Type A real-time traffic: This type of traffic has stringent packet-delivery delay requirements, but represents only a small portion of network traffic. The (worst-case) traffic pattern for this type is usually known in advance. Examples of this type of traffic are the synchronization and control signals in distributed real-time control systems. We assume the requested packet-delivery delay of 5 ms, the maximum packet size of 500 bits, and the maximum packet generation rate of 20 packets/second for each connection/channel of this type in the following example.

Type B real-time traffic: This type of traffic requires moderate packet-delivery delays, and forms a moderate portion of network traffic with prespecified (worst-case) traffic patterns. Examples of this type of traffic are packetized voice and periodic real-time data collection (e.g., sensor reading). We assume the requested packet delivery delay of 100 ms, the maximum packet size of 5000 bits, and the maximum packet generation rate of 20 packets/second for each connection/channel of this type in the following example.

In what follows, the FDDI timed-token network is compared with the buffered dual-ring network equipped with real-time channels with respect to network efficiency and the network's ability of supporting real-time communication.

Network Efficiency: For an FDDI timed-token ring network, in order to handle type A real-time traffic, the maximum token rotation time must be less than the minimum requested packet delay minus the ring latency $= 5ms - 0.4ms = 4.6ms$. The average token rotation time would thus be one half of the maximum token rotation time, i.e., 2.3 ms [12]. Thus, $(2.3ms - 0.4ms)/2.3ms = 82.6$ % of the link bandwidth is used to transmit packets, leading to a 200 Mbps $\times 82.6\% = 165.2$ Mbps network bandwidth for a dual-ring network.

By contrast, in a buffered ring network no time is wasted for token transmissions. Assuming the uniform distribution of source–destination distance and the shortest path routing, the average distance a packet needs to travel to reach its destination for a dual-ring network with 80 stations is 20 links. Thus, the average network bandwidth = 100 Mbps $\times 160$ links/20 links = 800 Mbps. One can see that by using buffers at intermediate stations, the network bandwidth becomes $800/165.2 = 4.84$ times higher than that without using buffers at intermediate stations.

Ability to Support Real–time Communication: We use the number of different types of real-time channels that a network can support as a yardstick in measuring the network's ability to support real-time communication. Let N_A and N_B denote respectively the number of type A and type B channels that a network can support. As discussed above, in the FDDI token ring the average token rotation time must not be greater than $2.3ms$, and during each token's rotation, $1.9ms$ can be used to transmit packets. The time needed to transmit a type A packet is 500 bits/100 Mbps $= 5\mu s$, and the time needed to transmit a type B packet is 5000 bits/100 Mbps $= 50\mu s$. Thus, the number of type A and type B channels that an FDDI dual-ring network can support must satisfy:

$$5N_A + 50N_B \leq 2 \times 1900 = 3600.$$

In the absence of type B traffic, the token ring can thus support up to $3600/5 = 720$ type A channels. However, the existence of type B traffic greatly reduces the network's ability to support type A traffic. For example, with 50 type B channels in the network, at most 220 type A channels can be established. In the absence of type A traffic, the maximum number of type B channels the network can support is 72.

The number of real-time channels that can be supported by a buffered network depends on the distribution of the source and destination stations. We assume the uniform distribution for this and ran simulations to determine the values of N_A and N_B. The number of successfully established type A channels vs. the number of type A channel establishment requests is

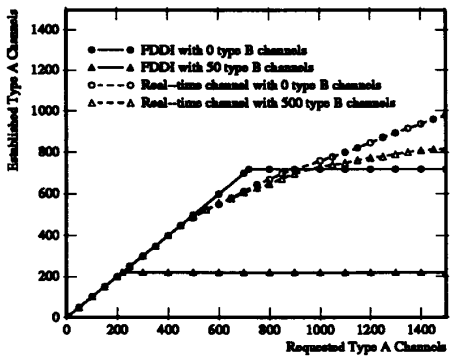

Figure 1: Network's ability to support real-time channels.

plotted in Fig. 1. In general, if the requested number of channels to be established is less than 400, all the requested channels can be established. As the number of requests for establishing channels increases, some of these requests will be rejected due to the increased load at certain links. The saturation point of the number of establishable type A channels is around 1000 in the absence of type B traffic.

From Fig. 1, it can be seen that without type B traffic, the timed-token network is a little superior in supporting type A channels to the buffered transmission network when the number of requested type A channels is between 400 and 720. However, this superiority disappears quickly as the number of type B channels increases. The reason for this is that a timed-token network has to be configured to support the most stringent real-time traffic, i.e., packets with the tightest delivery deadlines. All other real-time traffic is also guaranteed to meet this tight delivery delay requirement. This, in turn, results in inefficient use of network sources. So, the timed-token network is suitable for *homogeneous* real-time communication, while the buffered transmission with real-time channels is better suited to support *heterogeneous* real-time communication.

3 Hardware Support for Real-time Channels

To accommodate real-time channels, the physical network needs to support buffered transmission and deadline scheduling of packet transmissions.

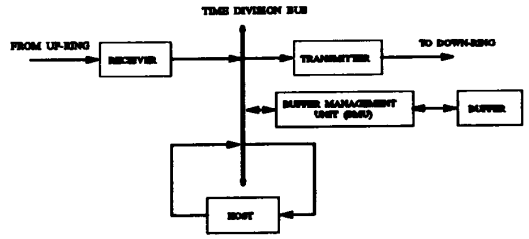

Figure 2: Architecture of the ring interface.

Buffered transmission has been widely used in multicomputer systems. Various switch architectures have been proposed [5, 6]. For a ring network, the switch is simpler than that for point-to-point connected multicomputers since there is only one incoming link and one outgoing link (a switch for a dual ring can be constructed with two single ring switches). The small number of incoming and outgoing links reduces the switch size and complexity of packet routing.

There are many advantages of using deadline scheduling for packet transmissions as discussed in the Introduction, but little has been done on the high-speed implementation of a deadline scheduler. The scheduler must be fast enough not to make itself a bottleneck of packet transmissions. Otherwise, all the advantages of using deadline scheduling will be lost.

In this section we will discuss the feasibility of hardware support for buffered transmission and deadline scheduling in high-speed local area ring networks. The proposed architecture of the ring interface is given in Fig. 2.

Because of the small number of incoming and outgoing links, one can use a high-speed time division bus to interconnect all components within each station. The bus cycles are interleaved into Receiver Write (RW), Transmitter Read (TR), and Host Write (HW) cycles. In other words, the bus cycles are composed of RW, TR, HW, RW, TR, HW, and so on. RW cycles are used to transfer packets from the receiver to the transmitter, the buffer, or the local host. HW cycles are used to transfer packets from the host to the transmitter or the buffer. Packets queued in the buffer are transferred to the transmitter using TR cycles. The operations of this interface are described below.

When a packet arrives at the receiver: The receiver does the serial-to-parallel conversion of the incoming packet. It buffers the packet until the destination address of the packet is received. Then,

- If the destination address matches the local host address, then the packet is written into the local host memory using the RW bus cycles.
- If the destination address does not match the local host address, then
  - if the outgoing link is idle, then the packet is directly forwarded to the transmitter using the RW cycles. In this case, packet *cut-through* is said to have occurred and a packet experiences the minimum switching delay.
  - if the outgoing link is busy, the packet is forwarded to the local buffer where it waits for later transmission.
- If the destination address indicates that the packet is being broadcast, the packet is forwarded simultaneously to both the host and the transmitter/buffer using the RW cycles. The receiver discards the packet if the source address of the packet matches the local host address.

When the local host wants to send a packet: The host sends out packets using the HW cycles. Similar to a packet from the receiver, if the transmitter is idle, the packet is directly forwarded to the transmitter. Otherwise, the packet is forwarded to the buffer for later transmission. In either case, the host is always able to output packets to the ring interface unless the buffer is full. Flow control can be easily implemented by requiring the host to check the status of the buffer before writing a packet into the ring interface.

When the transmitter finishes transmitting a packet: If the buffer is empty, the transmitter marks itself idle and waits for packets from the receiver or the host. Otherwise, it signals the buffer, and a waiting packet with the earliest deadline is transferred to the transmitter using the TR cycles. Notice that a packet can be forwarded to the transmitter before it is completely buffered. This reduces the packet buffering delay when cut-through was impossible initially but becomes possible later (before its complete buffering).

To make the above operations feasible, the buffer must be sufficiently fast. In the worst case, the buffer needs to accept packets from both the receiver and the host (using RW and HW cycles, respectively), and at the same time send packets to the transmitter (using the TR cycles). So the buffer bandwidth must be at least three times as high as that of the transmission

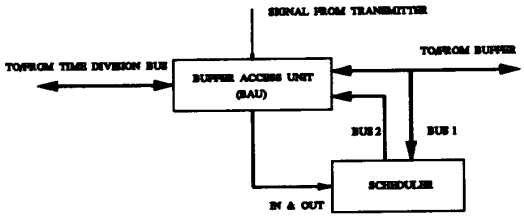

Figure 3: Architecture of the buffer management unit.

link, or 300 Mbps with the FDDI's 100 Mbps link transmission rate.

A key component of the buffer system is the Buffer Management Unit (BMU). The BMU provides two basic functions: (1) finding a free space in the buffer to accommodate an incoming packet, and (2) locating a packet with the earliest deadline in the buffer when the BMU is signaled by the transmitter.

The first function is basically a buffer bookkeeping function which can be implemented with several linked lists manipulated by dedicated hardware as discussed in [5, 13]. The hardware implementation of the second function, i.e., deadline scheduling, is relatively new. Thus, we will give a detailed account of scheduler design in the rest of this section.

The architecture of the BMU is shown in Fig. 3. When a packet arrives from the time-division bus, the Buffer Access Unit (BAU) will find a free space and store the packet in this space. A packet identifier which contains the starting address, the deadline, and the real-time channel ID of the packet is also sent to the scheduler if the incoming packet is a real-time packet. The scheduler maintains a priority queue for the identifiers of the real-time packets in the increasing order of their deadlines. When the transmitter signals the BAU to get a packet to transmit, the BAU simply reads the first item of the priority queue in the scheduler and fetches a packet from the buffer accordingly. If the priority queue is empty, a non real-time packet is transmitted on a FIFO basis.

The main function of the scheduler is the management of the priority queue. The packet-departure operation is quite simple. When the first item is read by the BAU, a single shift operation is enough to update the queue. The operation needed when a packet arrives, however, is more complex. The deadline of the incoming packet needs to be compared with those in the priority queue to find a position to insert the identifier of this new packet.

As mentioned before, a key design objective of the buffer system is its speed. So, the priority queue in-

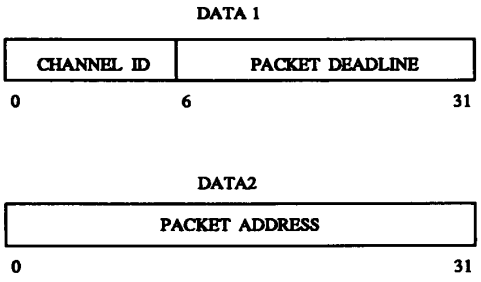

Figure 4: Data format

sertion must be implemented very efficiently. This can be realized by assigning a comparator to each storage cell and comparing all items stored in the priority queue with the new one simultaneously (e.g., associative memory). Needless to say, such an implementation is very expensive and the size of the priority queue must be kept under a certain limit.

Our design keeps the queue size equal to the maximum number of real-time channels to be accommodated by the ring interface. This is possible since the packets of a single real-time channel are always transmitted on a FIFO basis. Hence, the priority queue needs to hold at most one packet identifier for each channel. When this packet is transmitted, the identifier of the second packet of the channel (if there is one waiting in the buffer) gets inserted in the priority queue.

Given below is an example implementation of the above ideas. The details of its design and evaluation are presented in [14]. Suppose the interface of the scheduler to the BAU is composed of two buses, BUS 1 of width 32 and BUS 2 of width 33, and two signaling lines IN and OUT (see Fig. 3). When a real-time packet arrives at the BAU, the BAU will find a free buffer space for the packet and then assert the IN line to notify the scheduler of the arrival of a new packet. During the next two clock cycles, the BAU sends two data units DATA1 and DATA2 over BUS1 and then resets line IN. The formats of DATA1 and DATA2 are shown in Fig. 4. They contain the channel ID, the deadline of the packet, and the address in the buffer at which the packet is stored. The scheduler ignores all the following data transmitted over the bus (i.e., the packet itself to be stored in the buffer) until the IN line is asserted again.

The BAU reads the first item of the priority queue from BUS2. The last (33rd) bit of BUS2 is a valid bit which serves as a READY signal indicating if there are packets waiting to be transmitted. When the BAU is

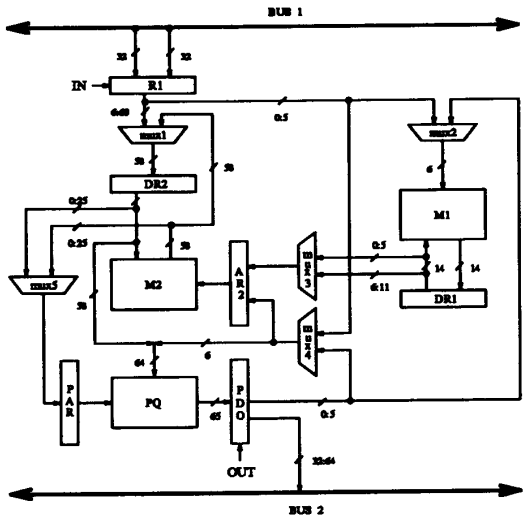

Figure 5: Scheduler data path

signaled by the transmitter and detects the READY signal, it reads the packet address from BUS2 and fetches the packet from the buffer to transmit. The BAU also asserts the OUT line for a couple of clock cycles to start the packet-departure operation of the scheduler.

The internal data path is shown in Fig. 5. PQ stores the priority queue. The arriving packet identifiers of a channel which already has a packet identifier in the priority queue are stored in M2. M1 stores the status of the channels. Its content in address i contains the head address (bits 0 - 5), tail address (bits 6 - 11) of channel i's FIFO queue in M2, and the 12th bit is set to 1 if the priority queue has a packet identifier of the channel, and the 13th bit is set to 1 if M2 has a packet identifier of the channel.

On arrival of a packet (see Fig. 6) the scheduler first stores the identifier of the packet, i.e., DATA1 and DATA2, in a register R1, it then checks if the priority queue (PQ) has a packet identifier of the same channel (this information is stored in memory M1). If not, the identifier is inserted in PQ. Otherwise, they are stored in memory M2.

On departure of a packet (see Fig. 7) the scheduler checks if M2 has a packet identifier of the same channel as that of the one just read by the BAU (again, this information is stored in M1). If yes, the packet identifier is fetched from M2 and inserted into PQ. Otherwise, it changes the contents of M1 to indicate that PQ no longer contains the ID of a packet belonging to the

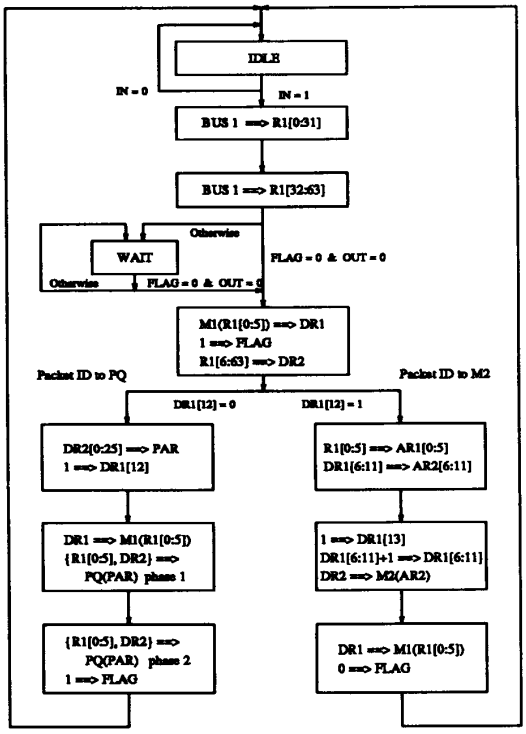

Figure 6: Flowgraph of scheduling an incoming packet

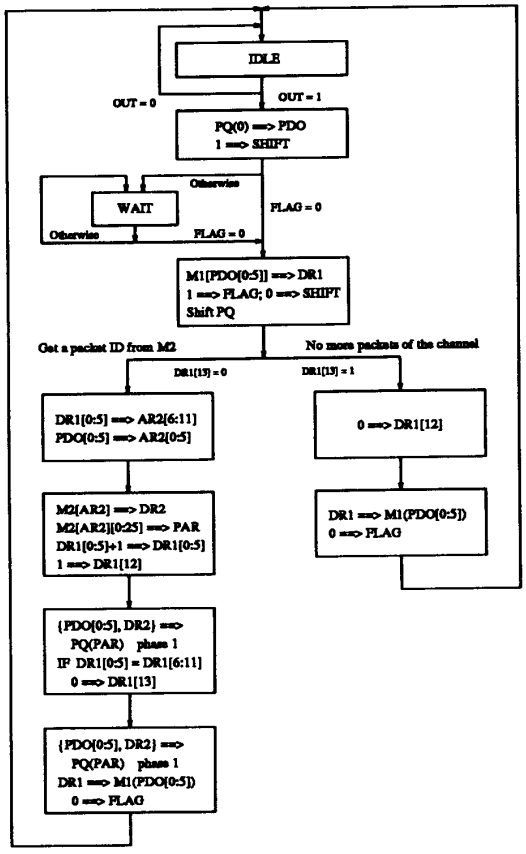

Figure 7: Flowgraph of outputing a packet

channel.

Insertion of a packet identifier into PQ is composed of two phases (thus needing two clock cycles): comparison and insertion. In the first phase, the deadline of the new packet (stored in the register PAR) is compared with the deadlines of all packets stored in PQ, and the position where the new packet should be inserted is found. The new packet is inserted in PQ in the second phase. The output of PQ is simply a shift right operation.

Since the arrival and departure of packets can happen simultaneously, care must be taken to avoid simultaneous requests for the memories and registers in the scheduler. This is achieved by adding two control bits FLAG and SHIFT.

As shown in Figs. 6 and 7, the FLAG bit allows only one operation (packet arrival or packet departure) to access the critical section. When the two operations try to enter the critical section at the same time, the access right is given to the packet departure operation by having the packet arrival operation check the OUT signal. The SHIFT bit controls the shift operation of PQ. As discussed above, when PQ outputs a packet, it shifts right. However, this shift operation cannot be performed when the packet arrival operation is in the critical section. Thus, the SHIFT bit tells PQ not to insert the new packet at the head of the queue; otherwise, it will be incorrectly shifted out.

The maximum scheduling time of the scheduler — defined as the time period between the arrival of a packet identifier at BUS1 and its insertion in PQ or M2 — is 12 clock cycles. With a not-so-high 12 MHz clock frequency, the scheduler is able to schedule 1 million packets per second. This speed is high enough to ensure the scheduler not to become a bottleneck in a 100 Mbps ring network.

4 Conclusion

We have discussed in this paper the advantages of using buffered transmission in local area ring networks. When it is equipped with real-time channels, the buffered transmission is shown to be superior to the usual token-controlled MAC protocols in both network throughput and the network's ability to support heterogeneous real-time traffic.

We have also shown the feasibility of hardware support for buffered transmission and real-time channels in high-speed local area networks. With the advances in VLSI technology, use of hardware to implement the switching and scheduling functions for buffered packet transmissions has become a reality.

Acknowledgment

The design of a fast deadline scheduler presented in the paper is a collaborative effort with A. Indiresan of RTCL. The authors also wish to thank S. Abraham of University of Michigan for his suggestions on the initial design of the scheduler.

References

[1] F. E. Ross, "An overview of FDDI: The fiber distributed data interface," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 7, pp. 1043 – 1051, September 1989.

[2] M. Teener and R. Gvozdanovic, "FDDI-II operation and architectures," in *proceedings of the 14th conference on local computer networks*, pp. 49–61, 1989.

[3] D. Dykeman and W. Bux, "Analysis and tuning of the FDDI media-access control protocol," *IEEE Journal on Selected Areas in Communications*, pp. 997 – 1010, July 1988.

[4] K. G. Shin and Q. Zheng, "Mixed time–constrained and non–time–constrained communications in local area networks," *IEEE Transactions on Communication* (in press), 1992.

[5] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, M. I. T. Press, Cambridge, Massachusetts, 1987.

[6] R. Suaya and G. Birtwistle, *VLSI and Parallel Computation*, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[7] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, vol. SAC-8, no. 3, pp. 368–379, April 1990.

[8] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," in *Proc. 11th Int. Conf. on Distributed Computer Systems*, pp. 300–307. IEEE, May 1991.

[9] Q. Zheng and K. G. Shin, "On the ability of establishing real–time channels in point–to–point packet–switched networks," *IEEE Transactions on Communication* (in press), 1992.

[10] Q. Zheng and K. G. Shin, "Fault-tolerant real-time communication in distributed computing systems," in *Proc. 22nd Annual International Symposium on Fault-tolerant Computing*, 1992.

[11] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[12] K. C. Sevicik and M. J. Johnson, "Analysis and tuning of the FDDI media access control protocol," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 6, pp. 997 – 1010, July 1987.

[13] Intel, *LAN Components User's Manual*, 1984.

[14] A. Indiresan and Q. Zheng, "Design and evaluation of a fast deadline scheduling switch for multicomputers," RTCL working document, December 1991.

Appendix

This appendix gives an algorithm for the calculation of the minimum guaranteed dalay bound of a real-time channel over a link. A real-time channel τ_i running through a link can be described as a 3–tuple (T_i, C_i, d_i), where T_i is the minimum packet inter-arrival time at the link, C_i is the maximum packet-transmission time over the link, and d_i is the packet delivery delay bound assigned to the link.

A set of channels $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, ..., n$, is said to be *schedulable* over a link if for all $1 \leq i \leq n$, the maximum queueing delay experienced by channel i's packets over the link is not greater than the requested delay bound d_i. The minimum guaranteed delay problem is stateed as follows:

Minimum Guaranteed Delay Problem:

Suppose $n-1$ channels, $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, ..., n-1$, are schedulable over a link. Given a new channel τ_n with the minimum packet inter-arrival time T_n and the maximum packet transmission time C_n, what is the minimum value of d_n such that all $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, ..., n$, are still schedulable?

Define $S = \cup_{i=1}^{n} S_i$, $S_i = \{d_i + nT_i : n = 0, 1, ..., \lfloor (t_{max} - d_i)/T_i \rfloor\}$, $t_{max} = \max\{d_1, ..., d_n, [\sum_{i=1}^{n}(1 - d_i/T_i)C_i]/(1 - \sum_{i=1}^{n} C_i/T_i)\}$, and $\lceil x \rceil^+ = n$ if $n-1 \leq x < n$, $n = 1, 2, ...$, and $\lceil x \rceil^+ = 0$ for $x < 0$.

We have the following Theorem giving the solution to the minimum guaranteed delay problem:

Theorem : Let $f(t, d_n) = \sum_{i=1}^{n} \lceil (t - d_i)/T_i \rceil^+ C_i$ and S be the set defined above with $d_n = C_n$. Then, $d_n = C_n$ is the solution to the minimum guaranteed delay problem if $\forall t \in S$, $f(t, C_n) \leq t$. Otherwise, the solution is $d_n = \max\{d^t : t \in G\}$, where $G = S \cap \{t : f(t, C_n) > t\}$ and d^t is computed as $d^t = C_n + k_f^t T_n + \epsilon_f^t + \epsilon_t^t$, with $k_f^t = \lceil (f(t, C_n) - t)/C_n \rceil - 1$, $\epsilon_f^t = f(t, C_n) - t - k_f^t C_n$, $\epsilon_t^t = t - C_n - k_t^t T_n$, $k_t^t = \lfloor (t - C_n)/T_n \rfloor$.

Interested readers are referred to [9] for the proof and more discussions about the theorem.