Advanced
Micro
Devices

The Effect of Local Buffer Memory Size on FDDI Throughput

by David Roberts

Publication #	*Rev.*	*Amendment*	*Issue Date*
16294	**A**	**/0**	**1/92**

The Effect of Local Buffer Memory Size on FDDI Throughput

by David Roberts

ABSTRACT

The effect of the size of local buffer packet memory on the throughput of FDDI adapters is examined. The analysis focuses on transmit buffer size and covers both the loaded and unloaded FDDI ring cases. A hypothetical EISA FDDI adapter design is used as a vehicle to demonstrate the throughput characteristics of different local buffer memory sizes. Calculations are done to size the buffer with respect to the latency of the EISA bus for the card in a target system configuration and with respect to the amount of effective bandwidth that the card can expect to receive. It is demonstrated that sizing a buffer based on these two methods results in poor throughput when the FDDI ring is loaded. As a solution to the performance problem, it is suggested that buffer sizing based on the TTRT value of the ring will result in an adapter that is capable of realizing high throughput. Finally, the AMD SUPERNET™ 2 FDDI chipset is examined to show that it is able to meet a diverse set of buffer sizing needs.

1. INTRODUCTION

The Fiber Distributed Data Interface (FDDI) is a 100 megabits per second (Mbps) local-area network (LAN) standard developed and standardized by the American National Standards Institute. The FDDI standard is designed to meet the greater LAN bandwidth requirements of the next generation of computing and information equipment. FDDI was developed to allow the rapid transmission of computer data as well as synchronous information, such as video, using a ring topology similar to the 802.5 (Token Ring) standard. FDDI allows for up to 500 stations separated by a maximum of 2 km of cabling distance between any two stations, with the total cabling length limited to 100 km. FDDI represents a full order of magnitude increase over the bandwidths offered by the 802 based networking standards such as 802.3 (Ethernet) and 802.5. The distances between stations and the high transmission bandwidth are made possible because the standard is based on fiber optic cable as its physical media (hence the “F” in FDDI). Schemes for transmission at the same data rates, with station separation distances limited to 100 meters, using shielded and unshielded, twisted pair, copper wiring are being investigated in the ANSI X3T9.5 committee and offer a lower cost medium when great distances are not needed.

One of the original FDDI design objectives was that the high raw bandwidth of FDDI should be translated into high effective throughput for nodes that use the network. To this end the FDDI access protocol was designed to insure that the network efficiency, defined as the ratio of obtainable bandwidth under high load to the 100 Mbps maximum bandwidth, would be high. Additionally, the designers tried to ensure that the obtainable bandwidth was fairly allocated among the nodes that were competing for it. Fairness is achieved through the FDDI access protocol.

Although the FDDI access protocol provides for the fair distribution of a large, obtainable bandwidth, nodes may or may not be able to realize the full throughput available to them, depending on their design. Poorly designed nodes can even limit the total network performance by destroying available bandwidth needlessly.

This paper will demonstrate how the local buffer memory size chosen for adapter cards can affect the throughput achievable by that card on the network. In particular, it will be shown that a transmit buffer size that takes into consideration the value of TTRT will lead to optimum performance of an adapter over both loaded and unloaded network and system bus conditions.

The following section defines some of the terminology and acronyms that are used in subsequent sections. Section 3 is short overview of FDDI, its basic access protocol, and some of the parameters which govern its performance. Section 4 describes some of the system parameters that must be understood when designing an FDDI node. Section 5 uses an FDDI EISA card as a design example and shows how the parameters affect the performance of the card under various system and network loads. The section demonstrates that a correctly tuned buffer memory will make the card more robust when additional stress is applied to it. In the final section, 6, the analysis from the previous sections is compared with the buffer memory size offered by AMD’s SUPERNET 2, FDDI chipset. The comparison shows that SUPERNET 2 is able to meet a variety of buffer sizing needs and methodologies.

2. DEFINITION OF TERMS

This section defines some of the acronyms, terms, and variables that will be used in later sections. Although many of these are defined in various other locations in this paper, this section provides a convenient reference point for those that may need it.

Publication #	Rev.	Amendment	Issue Date
16294	A	/0	1/92

AD: The access delay that a node may experience when it wishes to transmit on the network.

BW_{net}: The bandwidth of the FDDI network. This is a constant equal to 100 Mbps.

D: The ring latency of the FDDI ring.

DAS: Dual Attachment Station. An FDDI node which connects to both of the counter rotating rings in the FDDI network.

E: The efficiency that a system is able to achieve.

EISA: Extended Industry Standard Architecture. An extension of the original ISA bus. EISA features wider (32-bit) address and data busses as well as faster transfer protocols.

EBW_{sys}: The effective bandwidth of the system bus.

FDDI: Fiber Distributed Data Interface. A 100 Mbps ANSI standard LAN based on fiber optic media, a ring topology, and a token passing access protocol.

ISA: Industry Standard Architecture. The expansion bus architecture used by IBM PC-AT class personal computers.

L: The arbitration latency of a bus.

LAN: Local Area Network

N_{sys}: The number of bytes that the system bus can transfer during a given time interval.

SAS: Single Attachment Station. An FDDI node which attaches only to one ring of the FDDI network.

T_{4500}: The amount of time necessary to transmit 4500 bytes of data on the FDDI network.

T_F: The interval of time to refill a local buffer with data across the system bus.

THT: Token Hold Time. *The timer for this is described as a positive-valued down counter in this paper.* The value is computed as TTRT – TRT. If THT is a positive number, then the token is "early" relative to the negotiated TTRT value. In this case, asynchronous data may be transmitted by the node on this token rotation. If THT is negative, or zero, then there is no asynchronous bandwidth available to the node and the token must be passed after transmitting any synchronous data that the node may have.

TP: The throughput that a system is able to achieve.

TRT: Token Rotation Time. The actual time seen between two successive tokens for a given node. *The timer for this is described as a positive-valued up counter in this paper, which differs from other descriptions and many actual implementations. I feel that this makes the timer system a little easier to understand.*

TTRT: Target Token Rotation Time. The time negotiated by the nodes on the FDDI ring when it is initialized which is used to bound the time that a node may see between successive tokens. Its value is positive.

x: The size of a local transmit buffer memory.

802.3: A 10 Mbps IEEE standard LAN based on a bus topology and utilizing CSMA/CD as a medium access protocol.

802.5: A 4 and 16 Mbps IEEE standard LAN based on a ring topology and a token passing access protocol.

3. AN OVERVIEW OF FDDI

The following is a simple description of FDDI and some features of its access protocol. The following is not meant to be a complete description of FDDI. Rather, it just introduces some of the actions taken by the FDDI designers to insure a high efficiency for the FDDI protocol, and more important to this discussion, it introduces the concepts of Target Token Rotation Time, Token Rotation Time, and Token Holding Time. These will be important in the following sections.

FDDI has a basic topology of two counter rotating rings. One of the rings is used to carry data and the other provides redundancy in the event of a failure of a link between two stations. In the event of a failure, the two rings will wrap on either side of the failure into one large ring, restoring service. Stations which do not connect to both rings are called single attachment stations (SAS). They connect only to the primary ring through the use of a concentrator, which may, or may not, connect to the secondary ring. Although both the ring topology of FDDI and the token access to the ring are similar to 802.5, there are many differences in the actual access protocols between 802.5 and FDDI.

In addition to providing high, raw bandwidth, the designers of FDDI felt that it was important to insure that the bandwidth available on the network was usable by individual nodes, even when the network is busy. Bandwidth consuming collisions are avoided by using a token access protocol. A special packet of data, called a token, is passed from node to node around the ring. Nodes needing to transmit data hold the token when they receive it, called "capturing the token", transmit some data, and then pass the token to the next station downstream. This fundamental concept is the same for both 802.5 and FDDI.

In the 802.5 protocol, the frames that are transmitted on the network must be removed from the network by the transmitting station. In the 4 Mbps version of the standard, if a station finishes transmitting, it must wait, transmitting nothing, until the data that it put onto the ring has returned to it, full circle. Following this, it may then transmit the token to the next station. This scheme wastes

bandwidth because there is dead time on the ring while the station waits for its data to return to it. In fact, this problem was later corrected when the 16 Mbps version of the standard was developed.

In FDDI, the designers allowed for the early release of the token. Nodes must still strip their transmitted frames from the network when they receive them again, but they may transmit the token to the next node as soon as they are done transmitting their own data. The next node in turn can then capture the token and start transmitting immediately. Because of this, there is little dead time in a loaded ring.

FDDI was designed to carry synchronous data, such as video, in addition to computer data. The available bandwidth on the network is split into two types, synchronous and asynchronous. Nodes requiring a guaranteed bandwidth can use a synchronous transmission mode; on every token rotation the node captures the token and transmits for a guaranteed length of time. This time is negotiated among the nodes wishing to use the synchronous transmission mode, depending on their synchronous bandwidth requirements. Nodes that do not require a guaranteed bandwidth use the asynchronous transmission mode of the FDDI access protocol.

When an FDDI ring is initialized, a Target Token Rotation Time (TTRT) is negotiated among the nodes. The TTRT is the suggested time that the token should take to circulate around the ring when the ring is loaded. At most, the token will take 2TTRT to circle the ring (this will occur when the ring is previously quiet and nodes begin transmitting both synchronous and asynchronous data up to their bandwidth limits). The TTRT represents a bounded time quantum between transmissions that the nodes require for their bandwidth needs. As the token circles the ring, each node watches successive token arrival times to see if the token is early or late in arriving, as compared to the TTRT. The node measures the interval between token arrivals with a timer called the Token Rotation Timer (TRT).

Another timer, the Token Holding Timer (THT), is defined to be TTRT – TRT. If THT is a positive number, then THT represents the amount of time that the token is early, relative to the negotiated TTRT. When the token is early, the node has the opportunity to transmit asynchronous data. If THT is negative, then the token is late and no asynchronous data may be transmitted on this token rotation. The token must be passed immediately to the node's downstream neighbor. The node always transmits synchronous data whether the token is early or late. After transmitting any synchronous data that is ready, the node starts decrementing THT. While THT is greater than zero, the node can transmit asynchronous frames. When the THT expires by reaching zero, the node must pass the token to its downstream neighbor. If the node runs out of data to transmit before THT has expired, it should pass the token. If the node holds the token when it has no data to transmit, bandwidth will be wasted.

The TTRT, TRT, and THT are important to understand because they affect the Network Latency parameter described in the next section.

For more information about FDDI and its operating protocols, see [3] and [4].

4. THE IMPORTANT PARAMETERS

This section introduces some of the parameters that are important to an FDDI node's performance. These parameters assume an adapter card model, as shown in Figure 1. It should be noted that many "mother board" designs, although they don't use a separate adapter card, also fall into this model. It is assumed in this model that the host processor is the ultimate recipient for all of the data that is transmitted or received by the FDDI adapter. Therefore the data must be moved to or from the system memory whenever a packet is transmitted or received.

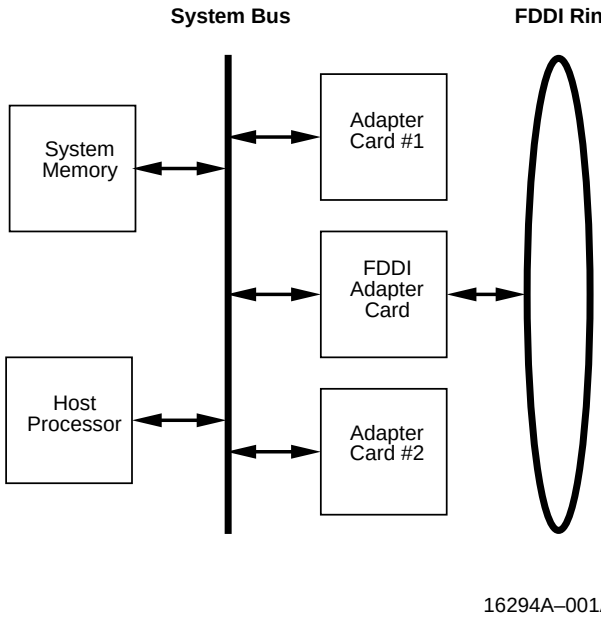

Figure 1. Adapter Card Model

4.1 Common Parameters

The two parameters that are most often used to help predict the local buffer requirements and system performance of an FDDI node are the peak bus bandwidth, and the bus latency. These are described below.

Peak Bus Bandwidth

The peak bus bandwidth is the peak theoretical rate of transfer between the system memory and the FDDI adapter. If the peak bus bandwidth is less than the FDDI maximum transfer rate, then the system bus will be the bottleneck for data transfer in the node and the node will

never be able to sustain the full FDDI bandwidth, even if both the FDDI ring and the system bus are unloaded. Further, whenever the node receives data from the network, it will be unable to move packets into the system memory faster than the incoming network reception rate. If a long stream of packets is directed at the node, the node may be overrun, leading to data loss.

Bus Latency

This parameter describes the length of time that an FDDI adapter must wait to access the system bus after it has decided that it will arbitrate for it. This is an important parameter because most networks are inherently asynchronous with respect to the host system. When an inbound packet arrives at the adapter and has to be transferred to system memory and when a transmit opportunity occurs cannot be predicted. When these events happen, the adapter must get access to system memory to either transfer the inbound packet in, or to transfer the outbound data out. The adapter must have enough buffering capability locally to overcome the bus latency during the time that it is waiting.

4.2 Frequently Overlooked Parameters

The following two parameters are also important in deciding how much buffer memory is needed local to an FDDI adapter. Although these parameters are often overlooked, they are very important in FDDI designs. The high data rate of FDDI along with the timed token access protocol demand that these parameters be understood by adapter card designers.

Effective Bus Bandwidth

Effective bus bandwidth refers to the actual transfer rate that a network adapter has into and out of the host memory over an extended period of time. If the bus is unloaded then the effective bus bandwidth is the same as the peak bandwidth. If other devices are using the bus over the period of time that the FDDI adapter is transferring data, then the effective bandwidth allocated to the FDDI adapter will be less than the peak bandwidth.

Some busses allow designated devices to have priority higher than that of other devices. If the bus arbitration scheme is such that once the FDDI adapter has arbitrated for the bus, that it can stay resident on the bus for as long at it would like (i.e. the adapter is not preemptable) then the effective bus bandwidth available to that card is equal to the peak bus bandwidth.

The calculation of effective bus bandwidth takes into account the above two parameters, bus transfer rate and latency, and a third parameter, transfers per arbitration cycle.

Network Access Delay

This parameter is similar to bus latency in that it defines the time difference from when the FDDI adapter knows that it has a packet to transmit and when the access protocol allows it to transmit. In the case of FDDI, this corresponds to the interval up to when the node is able to find a *capturable* token. The interaction between buffer memory size and this parameter will be shown in later sections. If a node is transmitting asynchronous data and must pass the token because the THT expires, the access delay is the amount of time it must wait before it can capture another usable token and begin transmitting again.

5. A CASE STUDY — EISA

The following section uses the Extended Industry Standard Architecture (EISA) Bus as a short example of how the various parameters that were presented in the previous section affect the amount of local buffer memory that should be included in an FDDI adapter design.

The EISA bus was developed by a consortium of PC vendors as a 32-bit extension of the Industry Standard Architecture (ISA) bus, which is used in IBM PC-AT class personal computers. The EISA bus supports 32-bit address and data paths, software configuration of cards at power up, and high speed transfer modes with peak bandwidths up to 33 MB/s. Additionally, it is upward compatible with older ISA cards. For more information about EISA, see [2].

5.1 Sample Configuration

During this example, imagine that we are responsible for the design of an EISA FDDI adapter card. The adapter card has been targeted as a server product and the model configuration contains the following cards:

1 802.3 Ethernet Bus Master

1 SCSI Bus Master

1 FDDI Bus Master, which we are designing

You may be wondering why all the cards are bus masters and none are DMA slaves. We will assume that all the cards in the system are high performance EISA cards that need to master the bus themselves to either manage advanced data structures like descriptor rings, in the case of the networking cards, or download command scripts, in the case of the SCSI card. This complex manipulation cannot be done by the card itself unless it is a bus master.

EISA is a preemptable bus and the EISA specification defines exactly how long a card can remain on the bus once it has been preempted. In the case of a bus master, the specification allows a card to continue to stay on the bus for up to 64 BCLKs after it is preempted. We will assume that since the networking cards are asynchronous in nature, they will always stay on the bus for 64 BCLKs after being preempted to try to reduce the chance of an under or overrun. We will also assume that since SCSI is a handshaken protocol and cannot overrun any FIFOs or local buffers, that the SCSI card will only remain on the bus for 32 BCLKs after it is preempted. This allows

more bandwidth to those cards that may need it when the system becomes busy.

Finally, we will assume that the FDDI card is a bursting bus master. In other words, the FDDI card is able to achieve the full EISA peak bandwidth of 33×10^6 bytes/s during the time that it is transferring on the bus. This is the peak bus bandwidth for this card, as defined in Section 4.1.

Calculation of Bus Latency

The following is a calculation of the FDDI card's bus latency parameter. This is the maximum time that the card can expect to wait for the bus, in this system. For the following calculations, the same assumptions used in the EISA specification to calculate system latencies will be used here, for those that are applicable. The assumptions are taken from Section 2.9.2 of the EISA Specification. These assumptions are:

- An 8 MHz EISA bus.
- The CPU releases the bus within 9 μs (32 BCLK periods plus 5 μs completion time for a locked cycle) after a preemption occurs.
- Bus Masters release the bus within 10.6 μs (64 BCLK periods plus completion time for the final cycle) after a preemption occurs.
- The DMA controller (programmed for BLOCK or DEMAND mode) releases the bus within 5.8 μs (32 BCLK periods plus completion time for the final cycle).
- A refresh cycle takes 1.3 μs.
- The CPU, DMA channels, and bus masters reassert their bus request signal immediately after relinquishing the bus after a preempt.

In our case, there are no cards that use slave DMA in our system; all the cards are bus masters. For the SCSI card, however, we will use the 5.8 μs number that the specification uses for DMA preemption time because the SCSI card stays on the bus for 32 BCLKs, like a DMA channel would. The card will arbitrate as a bus master, though, as it should.

The fact that all the devices will reassert their bus requests immediately after giving up the bus is indicative of a busy system. Thus the latency we are calculating is worst case for this particular configuration of cards.

The following table, Table 1, shows the sequence of events that occur after the FDDI card relinquishes control of the bus and rearbitrates for it. The order of arbitration is specified by the EISA specification. Simply, the arbitration is round-robin between three major groups: the CPU and bus masters, DMA channels, and refresh. In the CPU and bus master category, the CPU arbitrates every other cycle. The bus masters rotate through the remaining slots for this group. Refresh arbitrates once every 15 μs. The arbitration slots for the DMA slaves are shown in the table for completeness, but since there are no DMA slaves in the system configuration, the arbitration slots contribute nothing to the total arbitration latency of our FDDI card. For more information about the EISA arbitration scheme, see [2].

Table 1. EISA Arbitration Latency Calculation for FDDI Card

Device	Bus Hold Time (μs)	Total Time (μs)	Notes
FDDI Master	–	0	Release Bus
DMA	0	0	No DMA
Refresh	1.3	1.3	
CPU	9	10.3	
DMA	0	10.3	
Refresh	0	10.3	Total <15 μs
802.3 Master	10.6	20.9	
DMA	0	20.9	
Refresh	1.3	22.2	Total >15 μs
CPU	9	31.2	
DMA	0	31.2	
Refresh	1.3	32.5	Total >30 μs
SCSI Master	5.8	38.3	
DMA	0	38.3	
Refresh	0	38.3	Total <45 μs
CPU	9	47.3	
DMA	0	47.3	
Refresh	1.3	48.6	Total >45 μs
FDDI Master	–	48.6	Grant

The column labeled "Total Time" shows the accumulated time since the FDDI card released the bus until it gets it back. Since the card rearbitrates for the bus immediately after relinquishing control, we can see from the table that 48.6 μs is the maximum bus latency for the FDDI card in this system configuration. The bus latency could be much longer in a system that had more cards and was very busy, or much less in this same system if it was not busy at all.

Calculation of Effective Bus Bandwidth

Although the peak bus bandwidth of the EISA bus is 33×10^6 bytes/s, this bandwidth is only achievable while a card is on the bus. The bandwidth available to our FDDI card over a period of time, the effective bandwidth, is much less because of the demands placed on the bus by the CPU, the other cards, and the refresh mechanism. Knowing the effective bus bandwidth will help us size the local buffer memory of the FDDI adapter. The effective bus bandwidth of the FDDI card *in this configuration* can be calculated from the peak bus bandwidth, the bus residency time, and the bus latency.

The peak bus bandwidth is 33 x 10^6 bytes/s. With a clock rate of 8 MHz, this works out to one 32-bit word being transferred each clock cycle, which is, indeed, how the EISA burst protocol is designed to function. The bus residency time is 64 BCLKs after the card is preempted, which we will assume happens at the same time that the card takes control of the bus. Thus each time that the card starts transferring data, it gets preempted and must be off the bus within 64 BCLKs. If one word of data is transferred each clock while the card is on the bus, this amounts to 64 words, or 256 bytes, of data being transferred each time the card gains control.

Assume that the bus residency period is 10.6 μs, as the specification says it is for bus masters that are staying on the bus for 64 BCLKs after being preempted. Adding the bus residency time to the arbitration latency gives the whole arbitration cycle period. This is: 10.6 μs + 48.6 μs = 59.2 μs. Thus, every 59.2 μs, 256 bytes of data are transferred by the card. The effective bandwidth of the card is then: (256 bytes/arb cycle)/(59.2 μs/arb cycle) = 4.32 x 10^6 bytes/s, or 4.12 MB/s (1 MB = 1,048,576 bytes).

Since the calculation of the effective bandwidth is dependent on the arbitration latency, it is important to note that 4.12 MB/s is the effective bandwidth for the FDDI adapter *in this configuration and under this load*, only. If another card is added to the system, increasing the load, then the effective bandwidth of the adapter will decrease further. If the load on the system decreases because no more SCSI or 802.3 activity is present, then the effective bandwidth available to the FDDI adapter will increase.

5.2 Minimum Buffer Size to Prevent Transmit Buffer Underrun

The following sections describe the calculations necessary to size the buffer memory on the EISA card such that, during transmission of packets, a buffer underrun cannot occur. Two cases are examined: the first to overcome the maximum EISA bus latency the card can expect, and the second to overcome the limited effective bandwidth available to the card to transfer data. This discussion applies both to the synchronous FDDI transmission mode as well as the asynchronous mode.

Overcoming Bus Latency

Now imagine the following situation: the CPU has formatted an outgoing FDDI packet which it wishes to transmit onto the FDDI network. In preparation for transmitting the packet, the CPU has told the card to DMA the packet into the card's local buffer memory. The DMA engine starts and continues until the card's local buffer is full, and then stops. When a useable token is captured by the card, it starts to transmit the packet out onto the network, emptying the buffer memory. As the buffer memory empties, the DMA engine restarts, transferring the rest of the packet from the system memory to the card's local buffer. Given this scenario, how much buffer memory must be put on the card so that the transmission doesn't underrun when the DMA engine tries to restart by arbitrating for the bus (i.e. to overcome the bus latency)?

The question can be restated in the following way: how many bytes can the network transmit before the adapter can gain control of the bus to start refilling the buffer again? We know that the network data rate of FDDI is 100 Mbps, which is equivalent to 12.5 x 10^6 bytes/s, or 11.92 MB/s (1 MB = 1,048,576 bytes). During the adapter's arbitration latency of 59.2 μs, calculated in the previous section, the adapter will be able to transmit (12.5 x 10^6 bytes/s) x (59.2 μs) = 740 bytes. Thus, we need to have at least 740 bytes of data on the card, ready for the FDDI adapter to transmit, so that the card can rearbitrate for the bus and not have the packet buffer underflow during that time. *As we shall see, the amount of memory necessary to overcome the bus latency is the minimum amount of buffer memory necessary in any design.*

Overcoming Limited Effective Bandwidth

If the effective bandwidth available to the card to fill the buffer is larger than the emptying rate of the FDDI network, then the system bus latency is the dominant issue in determining the size of the local buffer memory. The system bus will be able to supply data faster than the network will be able to transmit it, and thus there will rarely be an underrun.

In practice, however, it is often very easy to show that the effective bus bandwidth available to the card can fall below the network transmission rate of 12.5 x 10^6 bytes/s. This is especially true when the card interfaces to a slow bus, or when a fast bus is heavily loaded. In the case of our EISA server example, the peak bus bandwidth is 33 x 10^6 bytes/s, which is more than double the network transmission rate. In spite of this, we demonstrated that when our configuration is loaded, the effective bandwidth available to the FDDI card drops to 4.32 x 10^6 bytes/s.

When the effective bandwidth available to fill the buffer memory is less than the network transmission rate, the card is unable to totally refill its buffer memory when it gains access to the bus. Thus, if the memory was sized simply to overcome the bus latency, the buffer might underflow during the next arbitration period. In this situation, the question becomes: how much buffer memory is needed to prevent underrun because of limited system fill rate?

The answer depends on how big the packet that we are transmitting is. If the packet size is less than 740 bytes, then we can already fit it into our 740 byte memory and we will never underrun when transmitting.

Additionally, since the peak bandwidth of the EISA bus is available to us *once we gain access to the bus*, if the card is able to download the rest of the packet to the local buffer in one arbitration cycle (i.e. rest of packet is ≤ 256 bytes), the packet will never underrun. In the worst case, after the card has been granted the bus, the 740 byte buffer memory will be empty. The card will have just transmitted the last byte from the buffer out onto the network. The card can now fill its memory at 33 x 10^6 bytes/s for up to 256 bytes. Since the 33 x 10^6 bytes/s fill rate is larger than the 12.5 x 10^6 bytes/s exit rate of the network, the amount of data in the buffer will increase during this time, and the packet won't underrun.

Remember, however, that as adapter designers, we would like to ensure that a maximum size packet will be able to be transmitted under a variety of circumstances without underruns. In the case of FDDI, we need to be able to transmit 4500 bytes without an underrun. If we wanted to take the easy way out, we would simply put a 4500 byte memory local to the adapter and transfer the entire packet to the buffer before we started transmitting data to the network. In practice this is inefficient because the amount of storage that is actually needed is probably less than 4500 bytes.

Imagine that we are about to transmit a 4500 byte packet out to the network. Let x bytes of data reside in the adapter local buffer, with $4500 - x$ bytes residing in system memory, waiting to be transferred to the adapter. This is the starting configuration of the system. Just as the adapter starts transmitting onto the network, the DMA engine in the adapter will start transferring the remaining bytes from the system memory to the local buffer memory. To size the buffer optimally, based on the effective bandwidth, the following condition should hold at the end of transmitting the packet: just as the network is about to transmit the final byte of the packet out onto the network, the DMA engine must have just transferred the last byte across the system bus into the local buffer. If this condition is met, and we live in an ideal world, then the adapter will be able to transmit the last byte without a problem and the packet will not underrun. Our task is to compute x given these conditions.

Solving for x takes four steps.

1. Compute the time that it takes the adapter to transmit 4500 bytes to the network. Call this time T_{4500}. Use the fact that the byte bandwidth of the FDDI network (BW_{net}) is 12.5 x 10^6 bytes/s = 100 Mbps/(8 bits/byte).
2. Compute the number of bytes that can be transferred over the system bus during time T_{4500}, given an effective bus bandwidth (EBW_{sys}) of 4.32 x 10^6 bytes/s. Call this amount of data N_{sys}.
3. Assert that $N_{sys} = 4500 - x$
4. Solve for x.

Step 1:

$$T_{4500} = (4500 \text{ bytes}) / BW_{net}$$
$$= (4500 \text{ bytes}) / (12.5 \times 10^6 \text{ bytes/s})$$
$$= 360 \times 10^{-6} \text{ s}$$
$$= 360 \ \mu\text{s}$$

Step 2:

$$N_{sys} = EBW_{sys} \times T_{4500}$$
$$= (4.32 \times 10^6 \text{ bytes/s}) \times (360 \ \mu\text{s})$$
$$= 1555 \text{ bytes}$$

Steps 3 & 4:

$$N_{sys} = 4500 - x.$$
$$x = 4500 - N_{sys}$$
$$= 4500 - 1555$$
$$= 2945 \text{ bytes}$$
$$= 2.88 \text{ KB (1 KB = 1024 bytes)}$$

Step 1 always gives a constant value because the maximum FDDI frame size and the FDDI transmission rate are fixed in the FDDI specification. The only independent variable in the calculations is the effective bandwidth of the system bus. Thus, these four steps can be reduced to the following simple formula:

$$x = (4500 \text{ bytes}) - [EBW_{sys} \times (360 \times 10^{-6} \text{ s})]$$

From the above calculations we can see that we must have at least 2.88 KB of local buffer memory in the FDDI adapter to be able to support the transmission of a 4500 byte packet without underrun, given an effective system bus bandwidth of 4.32 x 10^6 bytes/s. As you can see, this is a much larger number than the 740 bytes that are needed just to prevent underrun because of the system bus latency. In this case, the effective bus bandwidth is the dominant factor in computing the size of the buffer. Further, we shall see that 2.88 KB is simply the amount of memory that is needed to support *correct* operation under the above system load. *The following sections will show that overall network throughput will be extremely low and a larger memory is needed to support high throughput.*

Furthermore, it is important to note that the above analysis was for the transmit path only. The 2.88 KB value computed above does not take into account any memory that may be needed to buffer incoming frames. It can be shown, with an analysis similar to the one above, that the minimum amount of memory that is needed to buffer one incoming frame, assuming the same 4.32 x 10^6 bytes/s effective system bus bandwidth, is also 2.88 KB.

In the receive case, the system will have just transferred the 1555th byte of the incoming 4500 byte packet from the local buffer to the system memory when the last byte of the packet is copied from the network to the buffer by the FDDI adapter.

If the first packet is immediately followed by a second, however, the second packet will overrun the buffer and will have to be dropped because the DMA engine will not be able to empty the buffer fast enough to make room for the second incoming frame. To handle the case of many frames arriving in quick succession, the buffer size will have to be increased beyond 2.88 KB. To correctly size this buffer the expected reception traffic load is needed as well. This is left for another analysis. The rest of this paper focuses on transmission only.

Some readers may take exception to the fact that this example assumes that the EISA bus is always in use by all the devices that connect to it: the CPU, the refresh mechanism, and all three of the cards. In general the system would probably be quiet for much of the time. If the FDDI card were to access the bus randomly, it would find a very low latency on average and would be able to sustain close to the 33 MB/s peak bandwidth that EISA allows. In a server configuration, however, it is often the case that disk and CPU usage is closely coupled with network activity. Generally a client workstation generates a series of disk read or write requests over the network which leads to a lot of server CPU activity to process the packets, which in turn leads to a lot of disk activity to service the requests. This in turn leads to a lot more CPU activity and a lot more network activity as the requests are fulfilled. Although on average the system is quiet, it will become busy in a bursty manner which could lead to a lot of bus contention and all the effects which we have predicted in the previous sections for the duration of the burst. A client workstation that is performing a file transfer may experience similar activity.

Although the analysis assumes that a single 4500 byte packet will be transmitted, the argument also holds for multiple smaller sized packets whose total number of bytes is 4500. The analysis just computes the minimum buffer size to support a 4500 byte transmission. In fact, many network protocols, such as the Internet Protocol (IP), use default packet sizes smaller than 4500 bytes. IP does not place a limit on this, however, and many implementations are starting to use 4 KB as the packet size when FDDI is recognized as the underlying network. This reduces the per packet software processing overhead.

The analysis does not include any delay time in system bus transmission that may be associated between the additional packets, however. If multiple smaller packets are used, the buffer size needed to support a 4500 byte transmission should be larger than the amount that was calculated here to support the additional overhead time.

5.3 Throughput Calculations for Minimum Buffer Size

Given the buffer size calculated in section 5.2, 2.88 KB, this section examines the performance characteristics of the card on the FDDI side of the interface. Specifically, this section investigates the FDDI throughput that the card will be able to achieve when the network is both unloaded and loaded. Although the above discussion applied to synchronous transmission as well as asynchronous transmission, this section will only address asynchronous transmission. The throughput for synchronous transmission is guaranteed by the negotiation of synchronous bandwidth allocations among the nodes.

Ring Model and Assumptions

The following discussion assumes a configuration for the FDDI ring to which the EISA card is attached. A configuration of 20 single attach stations (SAS) on a 4 km fiber ring will be used. This configuration was chosen because it is identical to the “typical” configuration, used by Jain [1]. Jain states that these numbers are based on an intuitive feeling of what a typical ring would look like and not based on any survey of actual installations. This configuration could represent the floor of an office building. Twenty offices, using SAS, located on a 50 m by 50 m floor would require 2 km worth of cable or 4 km of fiber in a ring.

We will assume that a TTRT of 8 ms has been negotiated among the stations. Jain has found that this value provides for high ring efficiency and low maximum access delay over a wide range of ring sizes and loadings [1]. Additionally, we will assume that the speed of light through the fiber is 5.085 μs/km and that the delay that a bit suffers when passing through a station is on the order of 1 μs. These are the same assumptions used by Jain.

Given the ring configuration and the above assumptions, we are able to compute the ring latency. The ring latency describes the amount of time that it takes a bit of data to circle the ring and return to the transmitting station.

Calculating the ring latency is very straight forward. It is simply the sum of the travel time of the light representing each bit through the total fiber and the total bit delays through each node on the ring. In our case, the ring latency, *D*, is:

$$D = (4\text{ km}) \times (5.085\ \mu\text{s/km}) + (20\text{ stations}) \times (1\ \mu\text{s/station})$$
$$= 40\ \mu\text{s}$$

In section 5.2 we computed the minimum amount of local buffer that is required to insure that a 4500 byte FDDI packet will not suffer transmission underflow, given an effective system bus bandwidth of 4.32×10^6 bytes/s.

During the analysis, we assumed that to avoid underflow, the last byte of the packet would have to be transferred over the system bus just before it was required to be transmitted to the network. Thus, after transmitting 4500 bytes to the network, the FDDI adapter will be forced to pass the token to its downstream neighbor because it does not have any more data ready to transmit. If the adapter began to transmit another packet, it would immediately underrun. If it held on to the token, waiting for more data to be transferred over the system bus, it would waste FDDI bandwidth. In the following throughput analysis, therefore, we will assume that after each 4500 byte block is transmitted, the node must release the token back to the FDDI ring.

Unloaded Ring Throughput

Consider an unloaded FDDI ring, in the configuration given above, with our example EISA card as one of the attached nodes. At time t_0, our EISA node becomes active and wants to transmit a number of 4500 byte packets onto the network. The following figure, Figure 2, shows the sequence of events that occur. Time flows from the top of the figure toward the bottom. For reference, one other node on the ring (any node, it doesn't matter which) is shown. A token arriving at a node on the ring or being issued following a transmission is shown by a bold horizontal line. A bold vertical line indicates that a token is being held and that a frame is being transmitted by a particular node.

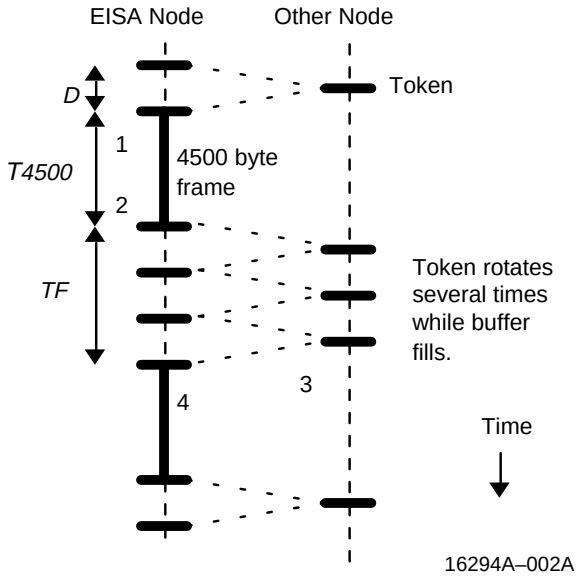

Figure 2. Unloaded Ring Event Diagram

1. Previous to time t_0, the token circles the ring, endlessly, with a rate of D = 40 µs, because the ring is unloaded. During this time the system fills the 2.88 KB buffer with the first part of the first packet that must be transmitted. At time t_0, the node captures a token and begins transmitting the first packet. At the same time, the DMA engine on the card is restarted to transfer the rest of the 4500 bytes of the first packet to the local buffer.
2. At $t_0 + T_{4500}$, the entire 4500 byte packet has been transmitted to the network. The local buffer memory is empty, thus the FDDI card reissues the token to the ring and begins to refill the local buffer with data from the second packet.
3. At the effective system bus bandwidth of 4.32×10^6 bytes/s, the 2.88 KB local buffer will fill in T_F = 681 µs. This time interval is much larger than the D = 40 µs that it takes the token to circle the ring. Thus the token will circle a number of times while the buffer is being filled. At time $t_0 + T_{4500} + T_F$ the buffer is full and the node waits for the token to pass by so that the node can capture it. This assumes that one 4500 byte packet is to be transmitted. If smaller packets are used, the token may be able to be captured earlier.
4. At this point, the token passes by and is captured. The time difference from the time when the node starts waiting for the token and the time at which it captures it is called the network access delay (See Section 4.2). When the ring is unloaded, the maximum access delay is D seconds, since the token travels around the ring at the maximum rate and the node has not transmitted for several token rotations. Again, one 4500 byte frame is transmitted and the token is released because there is no more data in the local buffer after the 4500 bytes have been transmitted.

The throughput achieved by the card in the above example is easily calculated. A single frame is transmitted every $T_{4500} + T_F$ seconds. If T_F is not an even multiple of the D, the ring latency, then the missing fraction is also added to T_F to make the total an even multiple of D. In our case, T_F is 681 µs, which is a little more than 17D. Thus, use 18D = 720 µs for T_F. T_{4500} is fixed by the network rate at 360 µs. The period of each transmission cycle is then 720 µs + 360 µs = 1.08 ms. The throughput is (4500 bytes)/(1.08 ms) = 4.2×10^6 bytes/s.

Although the FDDI bandwidth available to the card for transmission is 12.5×10^6 bytes/s, the throughput of the card is limited to the effective system bus bandwidth of 4.32×10^6 bytes/s. The card will only be able to achieve a maximum throughput equal to this rate. Given that we are being limited to 4.32×10^6 bytes/s, the 4.2×10^6 bytes/s that we are actually able to achieve is very efficient. To be more exact, it is (4.2)/(4.32) = 97% efficient. In the case of an unloaded ring, our example card performs very well.

Loaded Ring Throughput

Now let's see how our card performs when the ring is slightly loaded. Assume that as our node begins transmitting, one other node on the ring begins transmitting. Also assume that the other node is capable of utilizing all

the FDDI bandwidth, by itself. That is, once the other node captures the token, it will continue to transmit until its Token Holding Timer has expired. It will not release the token before it has to. The FDDI access protocol, because it is fair, will make sure that our card has the opportunity to utilize half of the 12.5 x 10^6 bytes/s FDDI bandwidth. The following figure, Figure 3, illustrates the sequence of events that occurs when the two nodes both start transmitting.

1. At time t_0 both our node and the other node start trying to transmit. Our EISA-based node captures the token first and starts transmitting its first 4500 byte frame. When it has transmitted 4500 bytes, it cannot transmit any more frames because the local buffer is empty, and so it releases the token and starts filling its buffer.
2. The other node wishing to transmit captures the token. The node sets its THT to the amount of time that the token is early. The amount, in this case, is TTRT – D – T_{4500} seconds. The node starts decrementing its THT and begins its transmission.
3. After TTRT – D – T_{4500} seconds, the other node's THT expires, and the node stops transmitting. It then releases the token to the ring.
4. The token arrives back at our EISA node. Unfortunately, the token is right on time. The Token Rotation Timer, which our node has been incrementing since the token last arrived, is exactly equal to the TTRT that was negotiated when the ring was initialized. Because the token is not early, the node cannot transmit asynchronous data at this time. Our EISA node passes the token.
5. The other node again captures the token, because it is early, and sets its THT to the amount of time that the token is early. In this case, the amount of time is T_{4500} seconds. Examine the diagram to see why this is so. The node begins transmitting data and decrementing its THT.
6. The other node's THT expires. It has transmitted 4500 bytes worth of data. It releases the token back to the ring.
7. Our node captures the token and transmits one more 4500 byte packet. It then releases the token because it has no more data ready to transmit. It begins to refill its buffer.
8. The other node now captures the token. It sets its THT for TTRT – D – 2T_{4500}. After transmitting for this amount of time, it releases the token once again.
9. Our EISA node captures the token and begins transmitting. After transmitting another 4500 bytes, it releases the token back to the network. It starts to refill its buffer.
10. The other node captures the token and transmits 4500 bytes, after which its THT expires and it releases the token.
11. Our node captures the token. It transmits its 4500 bytes and then releases the token. Some readers may notice this as a slight mistake in the analysis. In

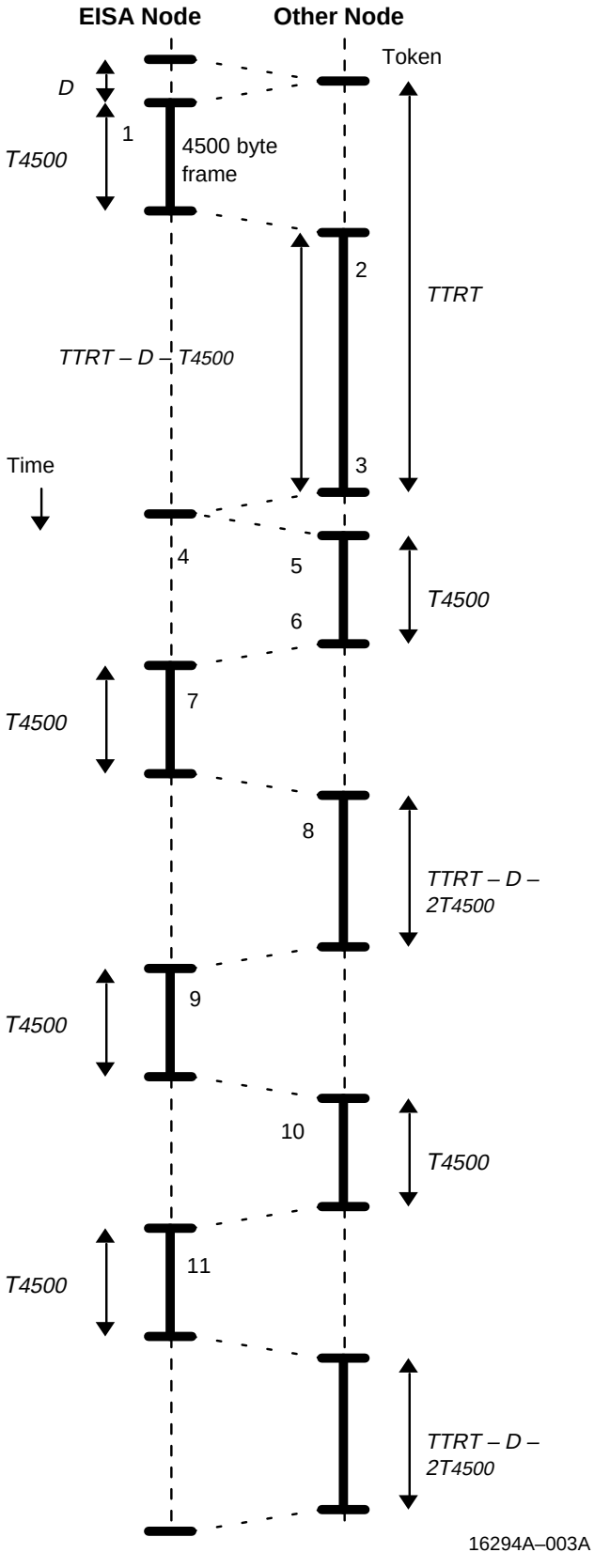

Figure 3. Loaded Ring Event Diagram

fact, our EISA node should not be able to capture the token at this time. The token has only been gone for $D + T_{4500}$ seconds. We know, from our analysis about minimum buffer sizes for preventing underrun, above, that during the $D + T_{4500} = 400\ \mu s$ token absence, our node can only transfer about 1700 bytes across the system bus. We know that we must have 2945 bytes in the buffer to prevent the underrun of the packet. Because of this, the node should pass the token because it is not ready to transmit. To simplify the analysis, however, we will allow the node to capture the token at this point. If the data was composed of several smaller packets, then the station might be able to capture the token, although it would not be able to transmit the full 4500 bytes. At the end of the analysis the throughput value that is calculated will be *higher* than the actual value, because we are both letting the node capture the token before it should be able to and are assuming that it is able to transmit the full 4500 bytes.

After steps 1 through 7, the sequence of steps 8 through 11 becomes a repeating cycle. The period of this cycle is TTRT + D + T_{4500}. During this time sequence, our EISA node transmits twice, at steps 9 and 11, for a total of 9000 bytes. The throughput of our node is:

$$\begin{aligned} TP &= (9000 \text{ bytes})/(\text{TTRT} + D + T_{4500}) \\ &= (9000 \text{ bytes})/(8 \text{ ms} + 40\ \mu s + 360\ \mu s) \\ &= (9000 \text{ bytes})/(8.4 \text{ ms}) \\ &= 1.07 \times 10^6 \text{ bytes/s} \end{aligned}$$

Given the 4.32×10^6 bytes/s maximum bus band-width, the efficiency of our FDDI card with one other node transmitting on the FDDI ring is:

$$\begin{aligned} E &= (1.07 \times 10^6 \text{ bytes/s})/(4.32 \times 10^6 \text{ bytes/s}) \\ &= 25\% \end{aligned}$$

Our card is only able to transmit one-fourth of the data that it should be able to! Remember, also, that this efficiency is *higher* than what the card should actually be able to accomplish because of the simplification we made in step 11 of the analysis. It is important to note that neither the network nor the system bus is limiting the card. The network access protocol guarantees that the card has the opportunity to transmit at a throughput of almost 6.25×10^6 bytes/s. We have previously shown that the effective system bus bandwidth is 4.32×10^6 bytes/s. Why is the card only able to attain a throughput of 1.07×10^6 bytes/s?

5.4 Problem Analysis

The Problem

The analysis given in section 5.2 is based on the assumption that as card designers all we are concerned with is the correct operation of the card during a loaded situation. That is, we simply want to insure that the card will not experience a transmit underrun while the system is busy. To this end, we designed the card with the minimum amount of outbound buffer memory to support the transmission of one 4500 byte frame without an underrun. When the FDDI ring was unloaded, we saw that our card was able to perform up to the limit set by the effective system bus bandwidth. The design was very efficient at converting both the available FDDI bandwidth and system bandwidth into data throughput. When the ring became loaded with just one other node, however, the throughput that our card was able to achieve dropped dramatically. Although both the effective system bus bandwidth and the available FDDI bandwidth were not the bottlenecks, the card was only able to achieve a throughput of about 25% of the effective system bus bandwidth. Let's do a short analysis to figure out why the performance was so low.

Consider our loaded ring example, above. After capturing the token and transmitting 4500 bytes, our card releases the token back to the network. The card's 2945 byte buffer is empty and it begins to refill with the next packet. When the buffer is full, the card waits for a capturable token to pass by. If the network is unloaded, then the time that the card will wait, the network access time, is less than or equal to D. If, however, the network is busy, then the card may have to wait a long time.

Jain has shown that in the worst case, a node may wait as long as $(n - 1)\text{TTRT} + 2D$ seconds, where n is the number of nodes that are actively transmitting on the FDDI ring at the time. This occurs when all the nodes are being "greedy" — transmitting until their THTs expire. It is important to note that the node is guaranteed to *see* a token in 2TTRT seconds. This must be the case for those nodes that have synchronous data to be transmitted. That token may not be *capturable* for asynchronous transmission, however, because the TRT value at the time may be greater than or equal to TTRT. The token at step 4 of the loaded ring diagram, above, is an example of such a token. Jain's equation gives the upper bound for the node to find a token that is capturable.

Some readers may have also noticed that in the case of only one active node, the equation predicts a maximum value of $2D$ for the access delay, as compared to the value of just D that was stated a couple of paragraphs ago. Why are these different?

The reason is that Jain's formula predicts the maximum delay with the assumption that the node transmits for its full THT time and then immediately tries to recapture the token. In this case, the first token that passes by the node will be uncapturable because the node transmitted for its full THT. Draw a small diagram like the previous ones, if this is still unclear.

In our case, however, our node is only transmitting 4500 bytes. Not only didn't we transmit for our full THT time, but we also let the token spin around the network sev-

eral times while we refilled our local buffer memory. Because of these two things, the next token to pass by the node is guaranteed to be capturable, leading to a maximum delay of just *D*.

In our loaded ring example, above, we can see that the maximum access time for our node is TTRT – $2T_{4500}$. Using the value for TTRT that we assumed above, we can compute the exact value of the network access delay for the situation that was described:

$$\begin{aligned} AD &= \text{TTRT} - 2T_{4500} \\ &= 8\text{ ms} - 2(360\ \mu\text{s}) \\ &= 7.28\text{ ms} \end{aligned}$$

Remember that this value only takes into account one other node on the FDDI ring being active. If many nodes are active, the network access delay can be much longer.

As was stated earlier, the time that it takes to fill the 2945 byte local buffer over the system bus is only 681 µs. After filling the buffer for 681 µs, our node could possibly wait for up to 7.28 ms – 682 µs = 6.6 ms before it gets the token again. During this time period, the system bus is not transferring any more FDDI data that may be stored in system memory! There is 6.6 ms of "dead time" on the system bus, with respect to the FDDI adapter (i.e. other cards could be using the bus, so it isn't totally dead). This dead time is caused simply by the fact that the buffer on the card is full.

The Solution

To achieve throughput that approaches the effective system bus bandwidth, the system bus needs to transfer data during the whole 7.28 ms of network access delay. When this happens, the bus will be transferring data constantly, both when the node has captured the token and when it is waiting, and the throughput of the card will be equal to the effective system bus bandwidth. To achieve this, the buffer on the card must be made larger.

After reading the above paragraph and examining Jain's formula for maximum access delay, one might believe that a card needs to have a buffer that is able to sustain a very large network latency. Further, the size of this latency is defined by the size of the ring, the number of nodes transmitting on the ring, and the negotiated TTRT value, all of which can vary greatly. Computing the size of this buffer would seem to be a difficult problem for a card designer. Fortunately, there is another constraint in the system — the THT.

When a node captures the token, it can only transmit for a limited time. The duration is the current THT value, which is equal to the amount of time that the captured token is early in arriving. At the most, the THT can be equal to TTRT – *D*. This occurs when the token is transmitted and no other node wishes to, or is able to, capture the token. The token simply returns to the transmitting node after suffering a rotation delay around the ring. Since *D* << TTRT, THT is bounded approximately by TTRT. Thus, the local buffer should be no larger than the number of bytes that can be transmitted to the network during TTRT seconds. If the buffer is larger than this, then the extra data won't be transmitted during the token capture, and the storage for it is wasted.

Given that the buffer should be larger than the size predicted by the underflow calculations, above, and smaller than the number of bytes that can be transmitted by the network during an interval of TTRT seconds, how big should the buffer be?

5.5 Calculation of Buffer Size to Assure Maximum Usage of Available Token Holding Time

To tune the size of the buffer, based on the TTRT value, we will use an argument that is similar to the one used to calculate the buffer size based on the effective system bus bandwidth. That is, we will assume that only part of the total amount of data that we wish to transfer will actually be present on the card when the card starts transmitting data to the network. The remaining portion of the data will be supplied by the system, across the EISA bus, during the transmission of the data on the network. The formula is just:

$$x = [\text{TTRT}/(80\text{ ns/byte})] - [EBW_{sys} \times \text{TTRT}]$$

The first term in the formula calculates the total number of bytes that we wish to transmit during the TTRT period. Eighty nanoseconds per byte is the rate of data transmission to the FDDI network. The second term computes how much data the system bus can supply to the card during the time that the node is transmitting for the full TTRT seconds. The difference of these two values is the number of bytes that must be on the card to prevent underflow of this block of data during transmission. The following calculation shows how much memory is needed for our example configuration, as described above.

$$\begin{aligned} x &= [(8\text{ ms}) / (80\text{ ns/byte})] - [(4.32 \times 10^6\text{ bytes/s}) * (8\text{ ms})] \\ &= 100{,}000\text{ bytes} - 34{,}560\text{ bytes} \\ &= 65.44 \times 10^3\text{ bytes} \\ &= 64\text{ KB} \end{aligned}$$

This value is much larger than the 4500 bytes that we might assume that we should use to size the buffer if we were simply trying to prevent underrun.

It is important to note that the above calculation is based on a TTRT value of 8 ms. This value was chosen for the example because Jain has shown that it is optimal over a great variety of ring sizes and configurations. Some FDDI rings use a higher value, however. Usually the

value is on the order of tens of milliseconds. If a TTRT of 20 ms is used, for instance, then the equation predicts a buffer size of 160 KB.

Of course, this discussion neglects the effect that software can play on sizing a buffer. Realistically, the protocol stack running on the host processor will impose some limitations on the amount of data that will be ready to be transmitted at any one time. Many TCP/IP implementations, for instance, use a sliding window protocol which limits the amount of unacknowledged data to eight IP packets. Although the default IP packet data size is 512 bytes, IP implementations using FDDI are starting to size their packets based on the FDDI maximum packet size of 4500 bytes. This reduces the number of packets in a transmission, and thus the total software overhead for a block of data. A common packet data size for these implementations is 4 KB. Using this packet data size, along with the eight outstanding packets, we can show that a local *transmit* buffer size of about 32 KB is probably all that is needed. The software will not have more than this amount of data to be transmitted at any one time, so sizing the buffer beyond this point will bring minimal performance gains.

It is important to note that other implementations and other protocols may have different requirements. The final size should take into account the present and future trends in protocols to size the buffer.

5.6 Throughput of TTRT Sized Buffers

The following section examines the throughput that might be achieved by our example EISA node in our example FDDI ring when a local buffer size of 64 KB is used.

Unloaded Ring Throughput

When the FDDI ring is unloaded, then our node will achieve performance that is virtually identical to what it achieved previously. When a whole frame has been transferred to the buffer, the node starts trying to capture a token. Since all tokens are usable and the network access delay is very small, the node will capture one almost immediately. Although the system bus has continued to transfer data to the buffer while the node was waiting for the token, the node waited for, at most, 40 µs. During that time, not much data got transferred, relative to the 64 KB maximum buffer size. The buffer is not nearly full.

After capturing the token, the node transmits all the data in the buffer and any more that the system can supply during that time. Eventually the buffer is emptied and the node lets the token go. The system bus continues to transfer data to the buffer again. The token is released early and the buffer memory is never completely filled to capacity. The system bus, however, is kept busy. It is constantly trying to fill a buffer that is always being emptied by the network. Since the system bus never stops transferring data, the throughput of the FDDI node is limited to the effective bandwidth of the FDDI adapter on the system bus. The adapter is able to achieve almost 100% of the data rate that the system bus is able to supply, up to the 12.5×10^6 bytes/s FDDI data rate limit.

Loaded Ring Throughput

In the case of a lightly loaded ring, where the network access delay is less than the amount of time required for the system bus to fill the 64 KB buffer, the same sequence of events occur as in the unloaded ring example. The system bus never stops transferring data to the card. Therefore, the adapter is able to achieve a throughput approaching 100% of the system bus effective bandwidth.

Consider, however, the case of a very loaded FDDI ring, where the network access delay is much greater that the interval of time required by the system bus to fill the 64 KB buffer. In this case, the system bus will fill the buffer while the node waits to capture a token and will stop transferring more data. The system bus will be idle with respect to the FDDI adapter while the node waits.

When the token is finally captured by the node, however, the node will transmit data for up to TTRT seconds, until it is preempted by the THT expiration. In this case, the FDDI bandwidth allocated to the node by the FDDI access protocol, based on the network load, has decreased to the point where it is less than the system bus effective bandwidth. The node is using all the FDDI bandwidth that has been allocated to it. The node is able to achieve 100% of the available FDDI bandwidth.

In this section, we have seen that when a large buffer size is used, sized according to the TTRT value of the ring, a node will be able to achieve 100% efficiency with respect to the system and the network. The throughput that the node is able to achieve is either limited by the bandwidth of the system bus when the network load is light, or by the allocated FDDI bandwidth when the network is loaded, whichever is smaller. The FDDI adapter has become essentially invisible between the system and the FDDI ring.

6. HOW DOES SUPERNET 2 STACK UP?

SUPERNET 2 is a second generation FDDI chipset available from Advanced Micro Devices, Inc. (AMD). SUPERNET 2 provides built in support of buffer memory sizes ranging from hundreds of bytes up to 256 KB. Expansion of buffer memory is easily accomplished because the buffer memory is implemented with external, standard, static RAM, available from a number of different memory vendors. This section shows that SUPERNET 2 is able to support all the common methods of buffer sizing, and in particular, sizing based on TTRT.

SUPERNET 2 is an ideal chipset for FDDI adapter card designs because it allows a designer to tailor a solution

to meet the appropriate market needs. Additionally, the chipset can easily provide a set of software compatible solutions that span many different bus architectures or target system loading environments. This results in reduced development effort of both hardware and software which, in turn, leads to a shorter time to market for a family of FDDI products.

6.1 Bus Latency Sizing

If a 256 KB buffer is used with SUPERNET 2 and half of the buffer is allocated to transmit data with the other half reserved for receive data, the chipset is able to support a maximum system bus latency of:

$$L = (256\ KB / 2) \times (1024\ bytes/KB) \times (80\ ns/byte)$$
$$= 10\ ms$$

Most system busses have maximum latencies on the order of hundreds of microseconds, so 10 ms is more than enough to cover even the most pathological operating conditions of these architectures. Note that this calculation also neglects the slight overhead of the claim and beacon frames which are permanently stored in the buffer memory.

6.3 Effective Bus Bandwidth Sizing

When effective bus bandwidth is used as the basis of computing the FDDI adapter buffer size the objective is to simply insure that underrun of maximum size FDDI packets is prevented, given that the effective system bus bandwidth is less than 12.5×10^6 bytes/s. From the buffer sizing formula that was presented in Section 5.2, we know that the size of the transmit buffer only needs to be 4500 bytes to support correct operation. This is size can easily be matched or exceeded by SUPERNET 2.

6.4 TTRT Based Sizing

The objective of TTRT based local buffer sizing is to ensure maximum performance over a wide range of system and FDDI ring loading environments. The default maximum value of TTRT is 165 ms. At the FDDI data rate of 100 Mbps, this corresponds to a buffer size of 2.06×10^6 bytes. Although this amount of memory is much larger than the 256 KB maximum buffer size supported by SUPERNET 2, realistic TTRT values will be much lower than this.

If we, again, assume that half of the 256 KB maximum buffer size of SUPERNET 2 is used for the transmit buffer, then we can show, using the equation in Section 5.5, that SUPERNET 2 can support TTRT values up to 10.5 ms. In this case we assumed that the effective bandwidth of the system was zero. If we assume an effective bandwidth of 4 MB/s for the system bus, then the 128 KB buffer will support a TTRT of up to 15.8 ms while maintaining 100% efficiency.

In Section 5.5, however, we stated that after a certain point the TTRT is not the dominant factor of the optimal buffer size. Rather, the amount of data that a protocol will have outstanding at any time can start to dominate. Realistically, this amount of data will be on the order of tens of kilobytes. Given this, 128 KB is probably plenty.

7. CONCLUSION

This paper discussed some of the factors that must be considered when sizing a local transmit buffer memory for an FDDI adapter card. An example based on an EISA FDDI adapter was used to demonstrate the various concepts involved.

It was shown that many of the simplistic methods of sizing a transmit buffer memory will cause great performance problems when the FDDI ring becomes loaded. The degradation is caused by the network access delay, which increases dramatically as more active nodes are added to the FDDI ring. These methods of buffer sizing do not take this effect into account.

It was shown that sizing the local buffer based on the negotiated Target Token Rotation Time allows the node to achieve the maximum throughput available to it. The node will have its throughput limited by either the system bus effective bandwidth or the allocated FDDI network bandwidth, whichever is smaller. The FDDI adapter will become an invisible connection between the FDDI ring and the host system.

Finally, it was shown that AMD's SUPERNET 2 chipset provides the flexibility to size a local buffer to meet many needs. Very important is the fact that SUPERNET 2 easily allows a designer to meet the buffer size requirements based on protocol and TTRT sizing.

SUPERNET 2 is the ideal solution for a family of products that span a number of different operating environments or price/performance characteristics. The flexibility offered by the buffer sizing ability allows a common core design to become the basis of many end products. The core hardware design can be reused with each product and the software similarity provided by the use of one chipset allows many designs to be completed quickly. This results in a great time to market advantage.

References

[1] R. Jain, "Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT," Proc. ACM SIGCOMM'90 Symposium on Communications Architectures and Protocols, September 1990, Philadelphia, PA, pp. 264–275.

[2] *EISA Specification, Version 3.0*, BCPR Services, Inc., 1989

[3] McCool, John F., "FDDI: Getting to know the inside of the ring", Data Communications, March, 1988.

[4] *A Primer to FDDI: Fiber Distributed Data Interface*, Digital Equipment Corporation, 1991.