Approximating the FDDI Synchronous Mode

David C. Etherton, Michael M. Johnson, George C. Polyzos

Computer Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

Abstract

We present protocol extensions supporting real-time traffic performance guarantees for Fiber Distributed Data Interface (FDDI) implementations lacking a synchronous mode of operation. We provide a brief description of the FDDI protocol, with an emphasis on the synchronous mode and network parameter tuning to meet real-time application demands, and then discuss hardware limitations of some first-generation FDDI adapters which make the synchronous mode ineffective and unable to support the required performance guarantees. By modifying the operating system kernel we can regulate the flow of information to the FDDI adapters and implement guarantees in higher-level software. Specifically, we added a two-tiered regulation facility which, at the lowest level, would place an upper bound on asynchronous transmission rate from the device driver buffers to the adapter. The next level up added the notion of standard and guaranteed performance system queues which user-level traffic could be directed into. Finally, a distributed user-level software mechanism was added to monitor and appropriately regulate the throughput from each adapter on the LAN in response to varying bandwidth requirements.

Keywords: real-time traffic, performance guarantees, network management, flow control, priorities, LANs.

1. Introduction

The Fiber Distributed Data Interface (FDDI) is an American National Standard [1], for a 100 Mb/s ring-based Local Area Network (LAN) architecture utilizing a token-passing protocol and optical transmission media. Copper-based versions of the system also exist, and are known as CDDI. We do not differentiate between the two since our discussion is limited to functionality above the physical layer. FDDI was designed as a second generation, high-speed LAN with parameters that are now considered appropriate for Metropolitan Area Networks (MANs), e.g., covering distances of up to 100 Km. A good overview of FDDI can be found in [2].

The FDDI protocol combines the statistical multiplexing capabilities of packet-switching with the guaranteed performance features of circuit-switching. In particular, the FDDI timed token rotation protocol provides support for two types of traffic: *synchronous* and *asynchronous*. The synchronous mode of FDDI provides guaranteed bandwidth and response time for applications that demand them. This is achieved through a reservation scheme. However, unlike true circuit-switching, where the unused bandwidth of dedicated circuits is wasted, unused synchronous bandwidth may be used immediately by asynchronous traffic. There is also an *isochronous* mode in the newer FDDI-II standard which provides circuit-switching capabilities. However, that is a true circuit-switching mode where reserved but unused capacity is wasted. Therefore, the isochronous mode is best suited to constant bit-rate applications, e.g., uncompressed voice communication, while the synchronous mode is ideal for variable bit-rate applications, such as compressed video.

Since FDDI was designed as a generic high-speed network capable of providing support for both bursty and unpredictable traffic and real-time applications with known characteristics and specific performance requirements, facilities for tuning its performance were made readily available. Both the response time of the protocol, i.e., the time within which network access is provided to a station requesting service, and the minimum guaranteed bandwidth available to each station can be controlled through manipulation of two FDDI parameters: the synchronous bandwidth allocation (ST) and the operational target token rotation time (*T_Opr*). Parameter tuning and the resulting performance of FDDI is discussed

---

This research has been supported by Digital Equipment Corporation and the UC MICRO program.

Address correspondence to George C. Polyzos, **polyzos@cs.ucsd.edu**, Tel.: +1-619-534-3508, FAX: +1-619-534-7029. David C. Etherton is now with Megatec Corporation, San Diego, CA. Michael Johnson is now with Livermore National Laboratories, Livermore, CA.

extensively in [3]. Proofs for the performance guarantees provided and related timing properties of FDDI are presented in [4] and [5].

Unfortunately, the synchronous mode of FDDI is considered an optional feature by manufacturers and it is not implemented in most FDDI equipment. The most obvious manifestation of this lack of support is hardware limitations. In particular, adapters do not possess separate synchronous and asynchronous transmit queues, but instead only a single queue into which all traffic is clumped. Since there is only one queue, synchronous packets must wait their turn behind previously queued asynchronous packets. This means that when a packet is given to the MAC sublayer it can request to use the synchronous mode of FDDI, but will only receive service after all previously received asynchronous packets have been flushed from the single traffic queue. This complicates, or even eliminates, the possibility of implementing the synchronous mode of FDDI, particularly without a real-time operating system. Instead, FDDI is now mainly used as a LAN backbone, typically interconnecting multiple Ethernets and other IEEE°802 LANs, without providing specific performance guarantees.

Our initial investigation quickly ascertained that adding a true synchronous mode with the FDDI adapters at our disposal was not a viable alternative because it would require modifications to the adapter's firmware. However, we soon found that by modifying the operating system kernel we could regulate the flow of information to the adapter and implement guarantees in higher-level software. Specifically, we added a two-tiered regulation facility which, at the lowest level, would place an upper bound on the information transmission rate from the device driver buffers to the adapter. The next level up added the notion of standard and guaranteed performance system queues which user-level traffic could be directed into. When it was time to transmit, the device driver would always look in the guaranteed queue for data and would transmit from the standard queue if and only if the guaranteed queue was empty.

In addition, a synchronous bandwidth allocation (SBA) mechanism and network-wide manager were designed and implemented, the definitions of which are hinted at in the FDDI Station Management (SMT) proposed protocol standard [6], yet left unspecified. To that end a distributed user-level software mechanism was added to monitor and appropriately regulate the throughput from each adapter on the LAN in response to varying bandwidth requirements. Finally, modifications to the standard socket calls were required in order to allow traffic to specify and use the two different modes of communications.

The remainder of the paper is organized as follows. In section 2 we present an overview of the FDDI timed-token protocol, together with the issues of parameter tuning and their effect on real-time traffic guarantees. In section 3, we present the essence of our scheme approximating the synchronous mode in the absence of appropriate system support. In section 4 we provide performance results obtained through a symbol-level simulator of FDDI which we developed. In section 5 we outline the implementation of the scheme, including the required operating system and device driver modifications, as well as user level software for managing the reservation requests and the resulting bandwidth allocations.

2. Overview of the Timing Aspects of the FDDI Protocol

In considering network support for real-time applications, it is important to recognize the requirements of these applications. Here we only consider a simple case which is very well suited to the guarantees offered by the FDDI synchronous mode. An application's requirements consist of the following two parameters: the application's maximum channel access time (CAT) requirement (specified as a rate, in units of 1/second), and the minimum service requirement (ST) per channel access (expressed in bits). These two characteristics are related to, but not equivalent to, the bit-rate of Constant Bit-Rate (CBR) applications. For example, any combination of CAT and ST can support a CBR application with a rate of ST/CAT, if appropriate buffering is available and a fixed (initial) delay of CAT is allowed. On the other hand, the specification of the application requirements through the CAT and ST parameters is more general and flexible.

The FDDI MAC protocol is based on the IEEE 802.5 token ring protocol. A station must capture the token before it can transmit data. However, unlike many other token-ring protocols, the station can place the token back onto the ring as soon as it has completed its transmission. This enhances efficiency, particularly in large rings where the propagation delay is significant. The most important FDDI parameter is the Operational Target Token Rotation Time, T_Opr, which determines the average token rotation time. During initialization, each station bids a Target Token Rotation Time (TTRT); the lowest value prevails and becomes the T_Opr for the ring.

Two system timers are maintained by each station, the Token Rotation Timer (TRT) and the Token Holding Timer (THT). The TRT measures the amount of time that has elapsed since the last token was seen by the station. Since tokens are consumed by stations transmitting data, and retransmitted later, a token will show up increasingly later as the amount of data being transmitted grows larger. The THT limits the amount of time a station may hold the token while it is transmitting data onto the ring.

Synchronous bandwidth is allocated and managed on a ring-wide basis by the Synchronous Bandwidth

Allocation mechanism (SBA). Synchronous frames are transmitted by a node immediately after the token has been captured. Then, if there is sufficient time remaining, asynchronous traffic can be sent up until it is time to relinquish the token. Each node has a predetermined allocation of synchronous bandwidth which is controlled by the Station Management (SMT). The total available bandwidth, both synchronous and asynchronous depends on the chosen network-wide *T_Opr*.

Individual station biding for a *T_Opr*, which will guarantee CAT, and reservation and relinquishing of synchronous bandwidth, is the responsibility of the SMT. At network initialization, each station is given a zero synchronous bandwidth allocation. When a station requires synchronous bandwidth, it sends a resource allocation request frame to the synchronous bandwidth allocation (SBA) manager. The manager determines whether or not to grant the request and returns a response to the requesting station. While the behavior of each station using synchronous bandwidth is defined in the standard, the actual allocation procedure used by the SBA manager is left unspecified. In addition, the SMT standard leaves the SBA manager selection scheme totally unspecified. Note that SMT protocols, e.g., neighbor identification, synchronous bandwidth reservation, etc., use asynchronous transmissions. Agrawal *et al.* [7] present and analyze several synchronous bandwidth allocation schemes.

During the bidding for *T_Opr* individual stations would bid a TTRT equal to half their desired CAT (so that the chosen *T_Opr* would satisfy their CAT requirement). FDDI parameter configuration is then straightforward and requires the following three steps.

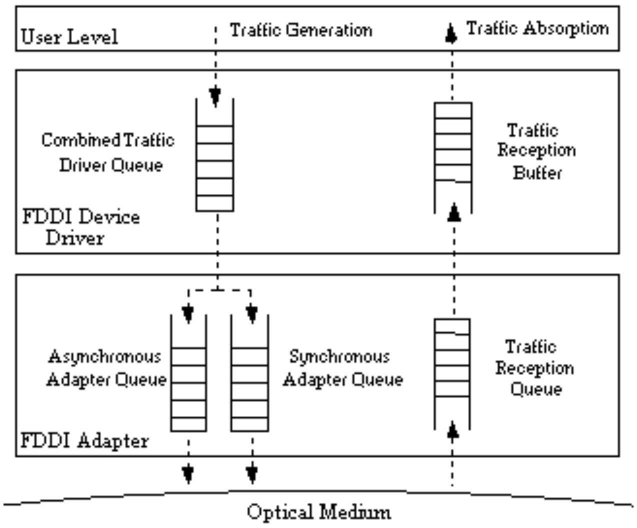


Figure 1: *StandardFDDI* Model Attachment Configuration

1. Determination of the minimum TTRT, $TTRT_{min}$, requested by a station on the ring (typically, $TTRT_{min} = 1/2\ CAT_{min}$).
2. Assignment of the operational token rotation value: $T\_Opr = TTRT_{min}$.
3. Determination of the total available synchronous bandwidth: TST $= T\_Opr -$ Overhead.

Figure 1 shows the standard FDDI attachment configuration; we will refer to it as the *StandardFDDI* model. User level applications generate and absorb traffic which, once passed down to the device driver level, is stored in a combined traffic queue. The adapter is responsible for notifying the device driver when it is starved for traffic. As traffic is passed to the hardware adapter, it is separated into two adapter queues according to the service requested: synchronous or asynchronous. The reception process is assumed to not be a bottleneck.

3. Approximating the FDDI Synchronous Mode in the Absence of Hardware Support

3.1. Overcoming Hardware Limitations

Many first generation FDDI adapters lack the necessary hardware support for managing the separate traffic queues. Without separate hardware queues, synchronous frames must be handled on an individual basis. This means that when a frame is given to the MAC layer it can request to use the synchronous mode of FDDI. However, since there is only one outgoing traffic queue on the adapter, synchronous frames are required to wait their turn in the queue behind asynchronous packets. Unfortunately, FDDI adapters are designed with large queues in order to achieve high performance.

Figure 2 presents the model attachment configuration for the *SingleQueueAdapter*. Under this scheme, the device driver merely forwards packets from the network layer system software out to the on-board adapter queue. The firmware on the adapter implements most of the FDDI protocol, as only it, being independent of the main system, is able to meet the real-time demands of the timed token protocol.

The most obvious solution to this problem is the redesign of the FDDI adapter. However, for users that must maintain hardware limited implementations of the FDDI protocol, the next best solution relies on relocating the queue management, normally located on the adapter, up into the device driver. The following discussion, applying to what we call the *SingleQueueAdapter* model, assumes a general split in functionality between the device driver and adapter, but does not get into specifics of particular implementations.

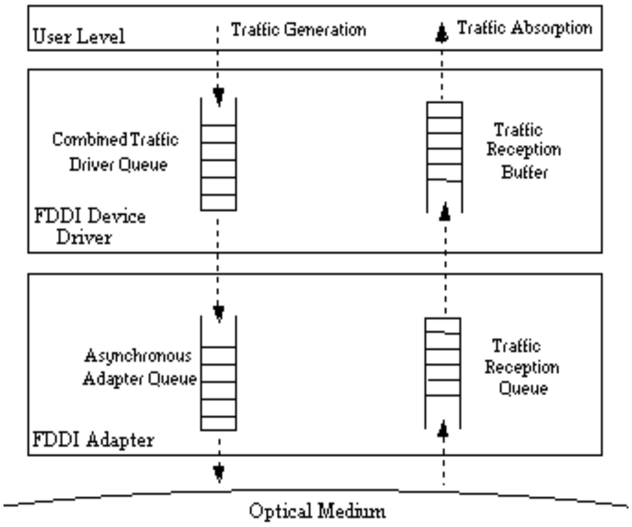


Figure 2: *SingleQueueAdapter* Model Attachment Configuration

Matters can be improved dramatically by implementing the device driver level software scheme indicated in Figure 3. Termed the *OracleScheme*, this configuration provides a means of segregating guaranteed and non-guaranteed traffic at the device driver level. The adapter portion of the model is identical to that presented in the *SingleQueueAdapter* model. Since traffic can now be prioritized above the adapter, the only functionality remaining to be provided is the control of how traffic is actually dropped down into the adapter. This is the responsibility of the bandwidth allocation mechanism, called the *Oracle*.

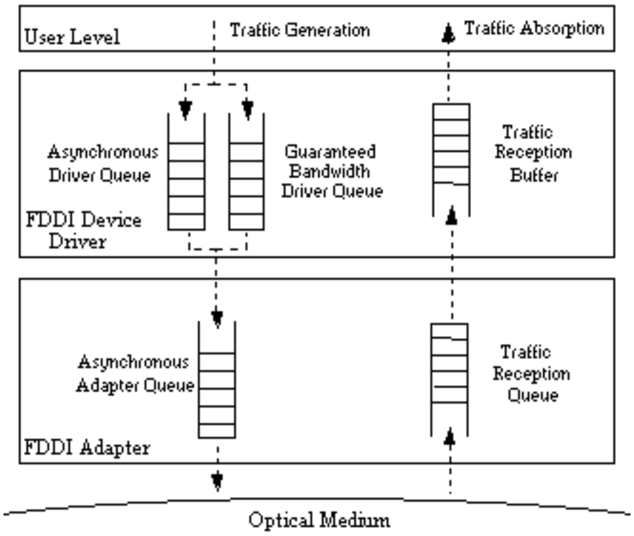


Figure 3: *OracleScheme* Attachment Configuration

3.2. Guarantees through Parameter Tuning and Adapter Starvation

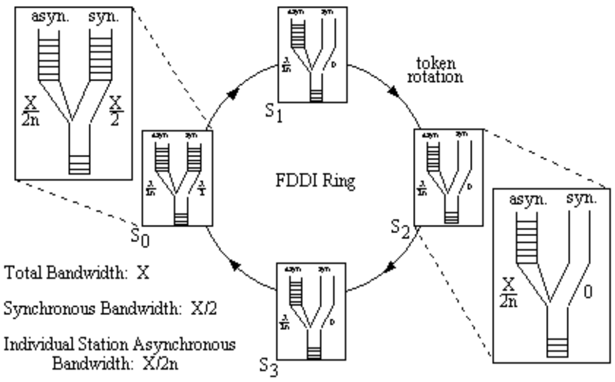
In order to meet synchronous mode guarantees under the *OracleScheme*, the *Oracle* must limit the rate at which traffic is allowed to flow into the adapter. Unfortunately, the *Oracle*, whose implementation resides totally within the operating system, is too far removed from the firmware of the adapter to adequately monitor the state of the ring and still provide a meaningful response time to network changes. This necessitates the enforcement of traffic regulation, which tends to penalize asynchronous traffic in order to have a better chance at meeting the constraints of synchronous traffic. However it allows the ring to operate normally when there are no real-time sessions in progress.

Once initial parameter arbitration has been completed by the *Oracle*, each station is guaranteed an amount of transmission time during each token rotation. Assuming there are no network failures and stations conform to the protocol, the issue of meeting synchronous mode guarantees becomes a problem of local station traffic restraint within the bounds of the arbitrated requirements.

3.3. *Oracle* Design and Ring Management

The *Oracle*, designed from the beginning with simplicity and ease of implementation as primary goals, manages routine synchronous bandwidth allocation requests at the station level. This includes initial reservation and reclaim of relinquished bandwidth. This design assumes a cooperative, error-free network environment. Each station is responsible for throttling the rate at which both synchronous and asynchronous traffic is allowed to proceed down into the single traffic queue of the adapter, a process termed adapter starvation. Since the throughput management facilities are implemented at the device-driver level, an *Oracle* process must be resident and executing on each machine in the ring to provide the traffic throttling functionality. A software agent, the *ManagingOracle,* manages bandwidth on a ring-wide scale. Together, individual station *Oracles* and the single, network-wide *ManagingOracle* provide the functionality required by Station Management (SMT).

Each station can intentionally starve its adapter from the device driver level to prevent more than a given number of bytes of data from being allowed to proceed down into the adapter within a given time period, defining an effective regulation rate. The *OracleScheme* is initiated by the first request for guaranteed bandwidth. The ring's *T_Opr* value is reset, if necessary, to match the CAT_{min} value. The total available bandwidth is then partitioned among all stations requesting guaranteed bandwidth. Each station has its bandwidth request normalized to the common *T_Opr* value. Any bandwidth left over after all guaranteed bandwidth has been distributed is evenly divided among all stations in the ring, including those which already share a portion of the guaranteed bandwidth. This extra bandwidth can be used by the stations for asynchronous traffic requests.



Figure˚4: Example Bandwidth Allocation with the *OracleScheme*

A *SlaveOracle* running on each machine is responsible for relaying bandwidth allocation and deallocation requests to the *ManagingOracle* at a known site, and also for transmitting the master's response back to the local client. The *SlaveOracle* also checks each local process owning guaranteed bandwidth to ensure that the process has not unexpectedly terminated without freeing the bandwidth; it will transmit a deallocation request to the *ManagingOracle* if the process is dead. The *ManagingOracle* must then redistribute the bandwidth among all stations on the ring after any successful allocation or deallocation operation, and inform each *SlaveOracle* about its current share of the bandwidth. The software should be designed to mimic the ring's original behavior when the guaranteed bandwidth facilities are not being used. When the last guaranteed traffic is deallocated, the ring is reset so that there are no limits on local output, and *T_Opr* is allowed to return to the adapter's default value. Assuming stations meet their traffic limits, then the synchronous traffic present will receive service within the specified guarantees of the synchronous mode of FDDI.

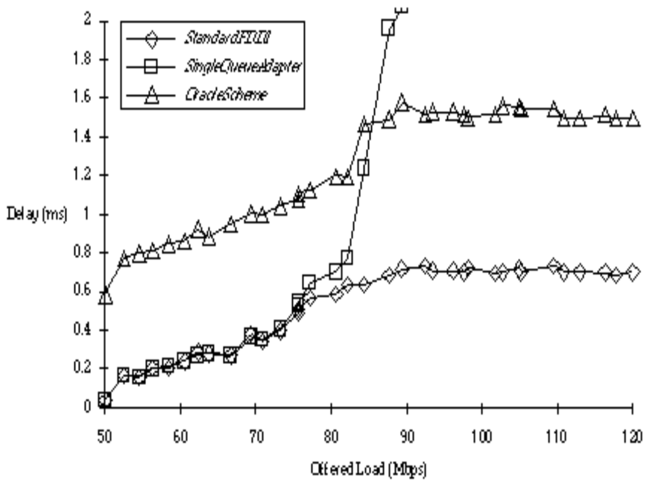
4. Simulation Results

In order to evaluate the performance limitations of the *SingleQueueAdapter* model, and demonstrate the operational characteristics of the *OracleScheme*, a symbol-level FDDI simulator was developed [8]. The following simulation results demonstrate the real-time traffic guarantees provided by the *OracleScheme* in relation to the synchronous mode service provided by *StandardFDDI*. To provide a fair comparison of the capabilities of each model, simulations were run using the same supplied traffic.

The network configuration employed is that presented in Table 1. Offered load ranges from the level presented by synchronous traffic alone, namely 50% of channel capacity, to far beyond maximum channel capacity, approaching 120%. According to the discussion of parameter tuning presented above, the *StandardFDDI* model was configured with a *T_Opr* of 1/2 CAT_{min}, i.e.., 600˚µs, whereas the *SingleQueueAdapter* and *OracleScheme* models were configured with a *T_Opr* equal to the specified CAT_{min}, namely 1,200 µs.

Table 1: Simulation Parameters

Number of Stations	15
Station Separation	100 m
Mean interarrival time of asynchronous frames (exponentially distributed)	12000 – 850 µs
Interarrival time of synchronous frames (fixed)	1200 µs
Asynchronous frame payload	500 Bytes
Synchronous frame payload	500 Bytes
Nominal latency per attachment	1.644 µs
Nominal station setup time following token capture	3.5 µs



Figure˚5. Impact of FDDI Adapter Model on Maximum Synchronous Frame Delay

Figure˚5 presents the maximum synchronous frame delay for each of the simulated models. The unbounded delay of the *SingleQueueAdapter* scheme is easily seen at approximately 80% of channel capacity. These results also demonstrate that the maximum frame delay of both the *StandardFDDI* and OracleScheme are bounded, even under extreme load conditions. Unfortunately, only the *StandardFDDI* model provides the necessary maximum channel access time guarantee of 1,200 µs. The *OracleScheme*, while satisfying the maximum frame delay requirement up to a load of 82%, suddenly jumps to a maximum channel access time of 1,500 µs.

This jump in maximum frame delay from 1,200 µs to 1,500 µs at an offered load of approximately 82% can be attributed to a condition we call token bypass. At loads below 82 Mb/s, the network has a sufficiently light load such that the *OracleScheme* is capable of providing station allocated bandwidth to the adapter in time for the next token rotation. However, at an offered load of around 82 Mb/s, the channel load is such that a synchronous frame passed to the adapter is required to wait behind previously queued asynchronous frames, which were unable to secure necessary bandwidth for transmission. The transmission of these asynchronous frames which were bypassed on the previous token rotation results in an additional delay for the synchronous frame just passed to the adapter.

An implementation of the *OracleScheme* could overcome this obstacle in a couple of different ways. The algorithm could restrict total bandwidth allocation to 82% of the channel capacity. Another possibility would be to gradually throttle off the asynchronous traffic to the adapter at offered loads approaching 80% of capacity. Both schemes would sacrifice obtainable channel utilization. However, this method provides a simple solution to the problem, requiring a simple change in bandwidth calculation by the *ManagingOracle*.

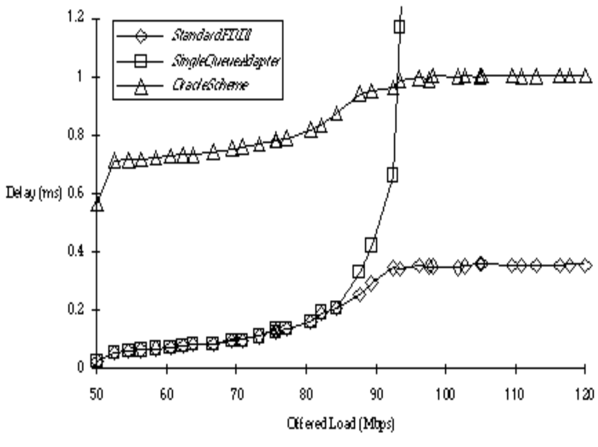


Figure 6: Maximum Synchronous Frame Delay with Adjusted Bandwidth Allocation

In solving this problem through a readjustment of bandwidth assignment, the *ManagingOracle* benefits from having complete knowledge of the total possible asynchronous bandwidth load. Since, at high loads, synchronous frames are waiting in the adapter queue behind asynchronous frames which were bypassed on the previous token rotation, the overhead of the extra wait incurred by synchronous frames should be equal to the total network-wide asynchronous bandwidth allotment. Therefore, in order to ensure that the synchronous bandwidth from all stations is capable of obtaining timely service, the total network-wide available asynchronous bandwidth must be reduced. The reduction in allowable asynchronous bandwidth should prevent the network load from exceeding 80% of capacity. With this modification available asynchronous bandwidth would be reduced from approximately 500 µs to around 300 µs in our example, due to the effects of token bypass.

Figure 6 presents the maximum synchronous frame delay for the default configuration after the bandwidth allocation adjustment. While once again the *SingleQueueAdapter* delay is unbounded, both *StandardFDDI* and the *OracleScheme* models provide service within the 1,200 µs channel access delay sought.

The average asynchronous frame delay is presented in Figure 7. For the *StandardFDDI* and *SingleQueueAdapter* models the slope of the curve does not begin to increase rapidly until an offered load of approximately 85% of channel capacity. The lowered asynchronous frame delay effects demonstrated by the *SingleQueueAdapter* model are a result of the fair treatment given to both asynchronous and synchronous traffic. Figure 7 shows the degree to which the *OracleScheme* over penalizes asynchronous traffic in order to meet synchronous traffic guarantees.

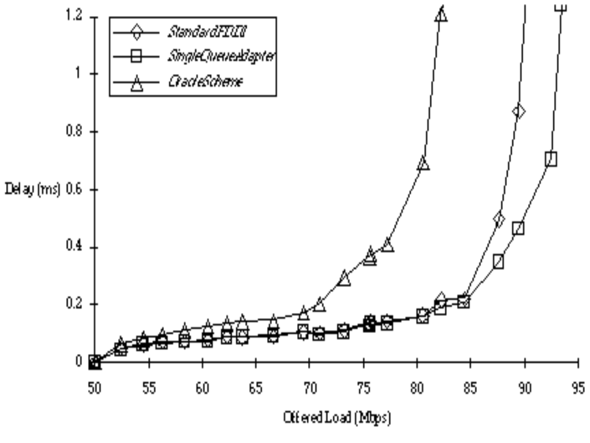


Figure 7: Impact of FDDI Adapter Model on Average Asynchronous Frame Delay

The *SingleQueueAdapter*, with its nonrestrictive use of available bandwidth, is best able to use the full network capacity. The *OracleScheme*, on the other hand, with its penalizing enforcement of traffic guarantees, leaves unused a slightly larger portion of the network capacity. Despite comparable traffic guarantees, the conservative nature of the *OracleScheme* results in a lower channel utilization than the *StandardFDDI* model.

5. *OracleScheme* Implementation Overview

The supervisor software implementing the OracleScheme is divided into a *SlaveOracle* and a *MasterOracle*. Every station on the ring runs a copy of the *SlaveOracle*; one station also runs the *MasterOracle*. The *MasterOracle* receives all bandwidth allocation and deallocation requests and determines the total bandwidth distribution. It then broadcasts the new global allocation to every *SlaveOracle* on the ring.

Each *SlaveOracle* serves as a relay between a local client and the *MasterOracle*. It also takes care of directly manipulating the local FDDI hardware to conform to the *MasterOracle*'s specifications. Finally, each *SlaveOracle* periodically checks every local process with guaranteed bandwidth to make sure it has not died, and deallocates its resources if it no longer exists. This prevents dead processes from consuming precious bandwidth.

The only other missing piece is a small amount of library code which simplifies the interface for opening and closing real-time connections. Taking full advantage of the facilities offered by the *Oracle* requires replacing the `socket(2)` and `close(2)` system calls with `rtsocket` and `rtclose`, respectively. The former is identical to the original `socket(2)` call with the addition of two new parameters specifying the maximum number of bytes sent per access, and the maximum acceptable CAT (in milliseconds). See [9] for further details and examples.

Incorporating adapter starvation and the *OracleScheme* into an existing asynchronous-only FDDI implementation required the following modifications.

Extension of default TTRT bid

In order to ensure the CAT requirements for *T_Opr* are not underbid by a station default adapter configuration value, upon initialization each conforming FDDI station is required to bid a default TTRT value equal to *T_Max*. to maximize available bandwidth. Since the lowest TTRT always wins ring initialization, this allows the TTRT request of the *ManagingOracle* to always succeed.

Network connection gatekeeper

Since network device access is gained through the network connection management interface, a mechanism must be established for the reservation and traffic priority assignment of individual network requests. This requires the implementation of a keyed access method for restricting unregistered network connections from gaining access to the synchronous traffic queue maintained at the driver level. For network connections requiring real-time traffic guarantees, a special driver level call should be provided in order to request, register, and gain access to the service guarantees provided by the synchronous mode driver queue.

Station *Oracle* inclusion

Within each network entity, an *Oracle* arbitration entity must be established, capable of receiving, interpreting, and managing guaranteed bandwidth allocation requests. In addition, a network-wide registration and arbitration entity must be implemented, the *ManagingOracle*, which is similar in function to the SMT SBA manager. This entity would be responsible for registering and arbitrating guaranteed, synchronous bandwidth characteristics of individual network connection feeds on a network wide basis. In order to receive the specialized and guaranteed service, a network connection feed must be individually registered with the authority, to include both worst case acceptable channel access time and bandwidth requirements.

Determination of operative token rotation

T_Opr must be selected to provide the necessary response time in support of the shortest requested CAT, $\mathrm{CAT}_{\mathrm{min}}$.

Network bandwidth allocation

Synchronous bandwidth is then allocated among individual stations based on a first-come, first-served request basis. Individual stations, which are allocated zero bandwidth by default, are allowed to contend for available synchronous bandwidth. However, the total synchronous bandwidth allocation must not exceed the maximum available bandwidth transmission time. Following complete arbitration of synchronous bandwidth, remaining bandwidth can be divided equally among all stations to support asynchronous traffic requests.

Management of local station traffic

During network operation, two frame queues are maintained within system level software, at the device driver level, one containing guaranteed-performance synchronous mode frames, and the other asynchronous frames. At predetermined rates, established through arbitration and maintained by the *ManagingOracle*, every *T_Opr*, a given amount of synchronous and asynchronous bandwidth is allocated to a station. Once the amount of appointed bandwidth for a given traffic class exceeds the demand of the next waiting frame of that class, the frame is allowed to proceed down into the single transmit queue of the adapter. Checks could be put into place to ensure the guaranteed service buffer pool does not grow beyond the registered level.

Device Driver Modification Overview

Code modification at the device driver level consisted of three major changes. First, several new calls were added to allow user-level software to get and set *T_Opr* and other ring parameters, get and set the maximum outbound flow rate (the regulation size, in bytes, and the regulation interval, in milliseconds), and set the internal software debugging flag. Second, the initialization code which causes a ring reset was modified to optionally substitute the new ring parameters supplied by the user—typically the *Oracle*—in place of the default values supplied by the adapter. Third, the primary output routine was modified to enforce the current regulation size and interval specified by the user. The routine was also automatically called by the kernel on a periodic basis while the regulation size was non-zero and there was still unsent traffic. Finally, when looking for packets to give to the adapter, the primary output routine always checked a new auxiliary system network queue (the guaranteed queue) before checking the standard one. This new queue was set up and properly fed by routines higher in the kernel, described next.

Socket Code Modification Overview

Sockets are the basic interface to network communication visible at the user level in a BSD UNIX-type environment. We added a new control flag to specify that all output on any given socket should be placed in the auxiliary guaranteed queue rather than the standard non guaranteed one. This control flag defaults to zero when a socket is created, so that programs unaware of the extensions (including existing system utilities such as ftp) will use the non-guaranteed service. A program must make a system call to explicitly set the flag, at which point all subsequent output will be routed through the faster queue. See [9] for a more detailed treatment of the modifications.

6. Conclusions

Lack of support for real-time traffic guarantees is becoming a major stumbling block in the evolution of interactive, time critical network applications. The Fiber Distributed Data Interface (FDDI), with a 100 Mb/s capacity, is capable of providing traffic guarantees through its synchronous mode. Unfortunately many first generation FDDI adapters lack the necessary hardware support for the synchronous mode. An evaluation of the effect of hardware limitations on the protocol’s performance was presented.

The *OracleScheme*, a device driver level scheme supporting real-time application performance guarantees was developed. Encompassing the, as yet undefined, Synchronous Bandwidth Allocation (SBA) mechanism and management functionalities required by the proposed SMT standard, the OracleScheme represents a complete management solution in support of real-time traffic. Despite imposed throughput limitations to non-guaranteed, asynchronous traffic, the performance guarantees presented to critical, real-time traffic are identical to those of full FDDI protocol implementations. A symbol-level FDDI simulator was designed and implemented and provided a tool for accurate performance evaluation.

Finally an overview of an implementation of the *OracleScheme* was also presented. Through the use of several interlocking software mechanisms we have created a ring-wide traffic regulation system which provides the required performance guarantees for real-time traffic.

References

[1] American National Standard, "FDDI Token Ring Media Access Control (MAC)," ANSI X3.139-1987.

[2] F.E. Ross, "An Overview of FDDI: The Fiber Distributed Data Interface," *IEEE JSAC*, vol. 7, no. 7, pp. 1043-1051, September 1989.

[3] D. Dykeman and W. Bux, "Analysis and Tuning of the FDDI Media Access Control Protocol," *IEEE JSAC*, vol. 6, no. 6, pp. 997-1010, July 1988.

[4] K.C. Sevcik and M.J. Johnson, "Cycle Time Properties of the FDDI Token Ring Protocol," *IEEE Transactions on Software Engineering*, vol. 13, no. 3, pp. 376-385, March 1987.

[5] M.J. Johnson, "Proof that Timing Requirements of the FDDI Token Ring Protocol are Satisfied," *IEEE Transactions on Communications*, vol. 35, no. 6, pp. 620-625, June 1987.

[6] Draft Proposed American National Standard, "FDDI Station Management (SMT)," ANSI X3.T9.5, Rev.°7.2, June 1992.

[7] Agrawal *et al.,* "Guaranteeing Synchronous Message Deadlines with the Time Token Protocol," Proc. 12th International Conference on Distributed Computing Systems, Yokohama, Japan, June 1992.

[8] M.M. Johnson, "FDDI Performance Guarantees for Real-Time Traffic in the Absence of a Synchronous Mode," M.Sc. Thesis, UCSD/CSE, La Jolla, CA, 1992.

[9] D.C. Etherton, "Approximating FDDI Synchronous Mode Guarantees Through Kernel Modification," M.Sc. Thesis, UCSD/CSE, La Jolla, CA, 1993.