Specification and analysis of the FDDI MAC protocol using systems of communicating machines

G M Lundy and I F Akyildiz* use a model to specify FDDI's MAC protocol, and analyse its safety and 'liveness'

A model designed for the specification of communication protocols called systems of communicating machines *is used to specify the FDDI token ring protocol, and to analyse its safety and 'liveness' properties. This model specifies each node as a finite state machine which has a set of* local variables. *With each transition is an* enabling predicate *and* action. *The predicates determine whether transitions may be taken, and actions alter the variable values as the transition executes. Communication between nodes is through* shared variables. *Our contributions include the specification of the basic FDDI protocol using a formal description technique; the use of this specification to analyse the safety and liveness properties of the network; the extension of the formal model to include the timing of transitions; and confirmation of the timing properties of the protocol.*

Keywords: MAC protocol, FDDI, protocol specification

Fibre Distributed Data Interface (FDDI) is a standard for an optical fibre network based on the token ring architecture which has been under development in recent years. It is the first major standard for optical fibre high-speed networks, and is expected to play a major role in future communications. At 100 Mbit/s its data rate is an order of magnitude improvement over current local area networks (LAN)[7, 8, 11, 12].

In this paper the basic FDDI protocol is formally specified and analysed using a model called *systems of communicating machines*. This model was designed for the purpose of describing and analysing communication protocols[1, 2], and has been used to model several well-known protocols. Each machine in the network is specified as a finite state machine augmented with local variables. Communication between machines is accomplished through shared variables. Each transition in the state machine has an *enabling predicate* and an *action*; these serve to unite the machine, the local variables, and the shared variables into a cohesive network.

Our primary contributions are in the specification of FDDI using a formally defined model, and in the analysis for safety and liveness. To accomplish this, the model definition was extended to include timing of transitions.

Much work has been carried out in the past decade on the formal modelling of protocols[3, 6]. A number of models have been suggested, including *Communicating Finite State Machines*, *Petri Nets*, *Estelle*, *LOTOS*, and several others. In this work we have chosen *systems of communicating machines* because it seemed to be an effective tool for the specification of this protocol,

Department of Computer Science, Naval Postgraduate School, Monterey, CA 93940-5000, USA
*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA
Paper received: 5 July 1991; revised paper received: 26 September 1991

providing flexibility as well as a formal basis for analysis.

In the following section we present the model *systems of communicating machines*. The FDDI is then specified using this model, and our analysis is given.

SYSTEMS OF COMMUNICATING MACHINES

The *systems of communicating machines* model was designed as a method for the formal description and verification of communication protocols[1]. It represents an effort to define a formal system which is useful in the description of network protocols, in their analysis or verification for correctness, and also in their conformance testing[2]. It is our belief that this model is a reasonable one for use in standards as well as for formal verification and testing; and that protocol implementors will find protocol specified in this way clear and unambiguous.

The following definition is an extension of the original definition to allow the explicit modelling of time.

A *system of communicating machines* is an ordered pair $\mathcal{C} = (M, V)$, where:

$$M = \{m_1, m_2, \ldots, m_n\}$$

is a finite set of *machines*, and:

$$V = \{v_1, v_2, \ldots, v_k\}$$

is a finite set of *shared variables*, with two designated subsets R_i and W_i specified each machine m_i. The subset R_i of V is called the set of *read access variables* for machine m_i, and the subset W_i the set of *write access variables* for m_i. The integers n and k are the number of elements (machines and variables) in sets M and V.

Each machine $m_i \in M$ is defined by a tuple $(S_i, s_0, L_i, N_i, \tau_i)$, where:

1 S_i is a finite set of states;
2 $s_0 \in S_i$ is a designated state called the *initial state* of m_i;
3 L_i is a finite set of *local variables*;
4 N_i is a finite set of transition names. Associated with each name is a unique triple (p, a, t), where p is an *enabling predicate*, a is the *action*, and t is a *time interval*. An action is a partial function:

$$a : L_i \times R_i \rightarrow L_i \times W_i$$

from the values contained in the local variables and read access variables to the values of the local variables and write access variables. The *time interval* t specifies an upper and lower bound on the time which a transition may be enabled before occurring. These limits are expressed in discrete units. If time limits are *not* specified, then default values of zero and infinity are assumed.
5 $\tau_i : S_i \times N_i \rightarrow S_i$ is a transition function which is a partial function from the states and names of m_i to the states of m_i.

In the original definition, the *time interval* was not included as a part of the transition name. If the time interval is taken to be $[0, \infty]$, then this definition is equivalent to the original.

Machines model the entities, which in a protocol system are processes and channels. The shared variables are the means of communication between the machines. Intuitively, R_i and W_i are the subsets of V to which m_i has read access and write access. A machine is allowed to make a transition from one state to another when the predicate associated with the name for that transition is true. Upon taking the transition, the action associated with that name is executed. The action changes the values of local and/or shared variables, thus allowing other predicates to become true.

The set L_i of local variables specifies a name and a range for each. As with the shared variables, the range must be a finite or countable set of values.

Let $\tau(s_1, n) = s_2$ be a transition which is defined on machine m_i (i.e. τ is the edge pointing from state s_1 to state s_2). Transition τ is *enabled* if the enabling predicate p associated with name n is true. The time interval t is measured from the point at which machine m_i is in state s_1 *and* predicate p is enabled.

Transition τ may be executed whenever the following three statements are true: (1) m_i is in state s_1; (2) the predicate p is enabled; and (3) the timing requirement (interval) is satisfied.

The *execution* of τ is an atomic action in which both the state change and the action a associated with n occur simultaneously.

If the following conditions hold, then transition τ *must* execute within the time interval:

1 A (finite) time interval on a transition is specified.
2 The machine is in a state from which the transition leads.
3 The transition is enabled throughout the time interval.

It is assumed that time passes at a constant rate within each machine.

The definition does *not* assume or require, however, that time passes (clocks tick) at the same rate from one machine to the next. However, in modelling protocols it is generally assumed that the rate of 'ticking' between clocks in different machines is within a specified margin of error.

A *system state tuple* is a tuple, or vector, of all machine states. For example, if a specified network has three machines (say 1, 2 and 3), which are currently in states 0, 2 and 4, respectively, then the system state tuple is (0,2,4).

The *global state* of a system consists of the system state tuple, *plus* the values of all variables.

The *system state* consists of the system state tuple, plus an indication (listing) of the enabled outgoing transitions. Thus a system state provides more knowledge of the system that a system state tuple, without providing the complete global state.

These definitions are useful in the analysis of protocols such as those which appear later in this paper. Further discussions of the model can be found elsewhere[1, 2].

FDDI PROTOCOL SPECIFICATION

The protocol is described briefly in the following subsection, and then the formal specification is given.

Overview and formats

FDDI is a *timed token ring network*, i.e. the nodes are connected in a series of point to point links forming a cycle; the right to transmit is controlled by passing a *token* (special message) from station to station; and the time for holding the token is strictly controlled, so that every station is guaranteed the right to transmit a specified amount of data within a certain time limit.

The standard specifies that *two* unidirectional rings, with data flow in opposite directions, will be included. The second ring provides redundancy in case of failure. The standard provides for at least three types of station (single attached, dual attached, and concentrators); however, for the purpose of the MAC specification in this work, all nodes will be treated as identical.

Message types: there are two primary types of traffic: *synchronous* and *asynchronous*. Synchronous traffic is time critical; it must be transmitted within strict time limits. Examples are voice traffic or real-time data. Asynchronous traffic does not have strict time constraints. Upon receiving the token, each station is allowed a specified time for transmission of the synchronous traffic. This time is determined when the ring is initialized, and may vary from station to station. The sum total of synchronous time allotment for all stations must be less than the TTRT, the *target token rotation time*. Thus, the protocol is able to guarantee every station a minimum amount of transmission time with a maximum wait between transmissions.

Asynchronous traffic may only be transmitted when the token is 'early', or ahead of its target rotation time.

Node structure: each node on the network has functions which may be divided into three categories: the *physical* functions, or physical layer; the *media access control* (MAC), which interfaces to the physical; and (SMT) *station management*.

The physical layer is further divided into two sublayers called PHY and PMD (*physical medium dependent*). The PHY interfaces with the MAC layer and the PMD; the PMD interfaces with the medium (fibre).

The physical layer is concerned with the actual connection of fibre to the station, the transmitting and receiving hardware, the type of fibre and physical connectors. It also specifies the encoding of the data received from the MAC into signals, timing requirements, etc. The PHY receives *symbols* from the MAC layer, encodes and transmits them (through the PMD) to the PHY at the next station, which decodes them and passes them to the MAC at that station.

The medium access control layer implements the timed token ring protocol. One the sending side, it receives a message (sequence of characters) to transmit as input. It encodes these into symbols - characters of data or special control symbols - and groups the symbols into messages (frames) for transmission. The frames are passed to the PHY, symbol by symbol, for transmission. The receiving part of the MAC receives symbols from PHY, and regroups them into messages. The data in these frames are then passed on to the application. The MAC is also responsible for token passing, which controls the right to transmit on the medium. Figure 1 shows the relationship between PHY, MAC and the user or application.

The station management layer is concerned with network initialization, recovery from ring failures, and other control functions.

The complete FDDI standard includes a document for the PMD, PHY, MAC and station management layers.

The specification and analysis of this paper is concerned with the medium access control (MAC) layer. The physical layer (PHY and PMD) is also modelled; this is necessary because of the interface between the two layers. However, the physical part of our specification is an abstraction of the actual physical layer, meant to retain key properties (for our purposes) and eliminate unnecessary details.

The set of symbols consists of data values from 0 through 15, and some special control symbols. Each symbol is coded a five-bit string. The symbol set used in this paper is:

{I,J,K,T,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}.

The symbol I is the 'idle' symbol; and the symbols J, K and T are used as delimiters. The data symbols are from 0 to 15 (decimal), where 'A' through 'F' represent the (decimal) data values 10 through 15. The FDDI standard has four additional control symbols (Quiet, Halt, Set and Reset) which were not included in this specification.

Token Format: the token consists of four fields: preamble (PA), starting delimiter (SD), frame control (FC) and ending delimiter (ED). Table 1 depicts the fields used and their values. The preamble in the FDDI standard has a variable number of 'I' symbols; the originating station transmits 16 'I' symbols, and other stations may shorten it or lengthen it to meet physical layer clocking requirements. However, in this paper, for the sake of brevity, a single 'I' symbol is used as a preamble. When written as symbols, the token consists of the seven symbol sequence [I,J,K,0,0,T,T].

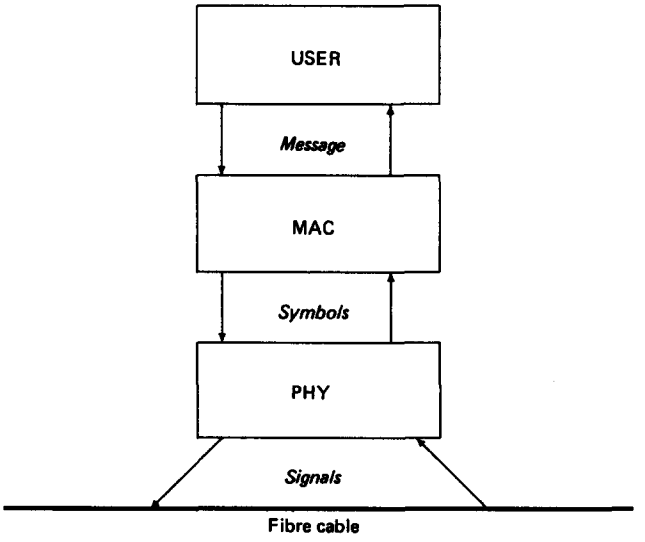

Figure 1. FDDI *structure of the layer*

Table 1. Token and frame fields

Name	*Ab*	*Values*
Preamble	*PA*	*I*
Starting delimiter	*SD*	*JK*
Frame control	*FC*	00(*token*) 01(*synch. frame*) 02(*asynch. frame*)
Destination address	*DA*	
Source address	*SA*	
Information	*INFO*	sequence of data symbols
Frame check sequence	*FCS*	(error code)
Ending delimiter	*ED*	*T*
Frame status	*FS*	0 or 1

Frame format: a data frame consists of the following fields:

[PA,SD,FC,DA,SA,INFO,FCS,ED,FS].

The additional fields are destination and source addresses (DA,SA); information (INFO); frame check sequence (FCS); and frame status (FS). The FC field indicates whether the frame is a token or (data) frame. Table 1 shows which symbols are used in each field.

Formal specification

The specification of the FDDI network formally as a *system of communicating machines* consists of (1) the specification of the network stations, (2) the shared variables through which they communicate, and (3) the designated initial state of the system.

Structure and shared variables

The overall structure of the network and the relation between each machine and the shared variables is shown in Figure 2. Each machine shares one variable with its upstream neighbour. This variable is of the type *buffer*, where *buffer* is specified by:

type *buffer*: array[1 . . MFL + 1] of symbol;

where *symbol* may be any of the symbol values listed in the previous subsection, or the empty value, denoted by 'Ø'. The value of 'MFL' is the maximum frame length. (The maximum frame length in the standard is 9000 symbols, which is 4500 bytes.) Each machine refers to the incoming shared variable as *inbuf*, and the outgoings as *outbuf*.

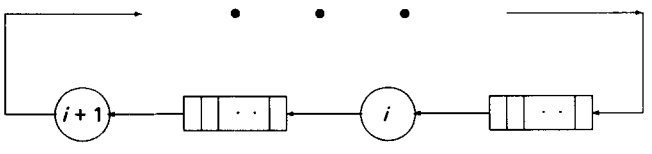

Figure 2. Modelling the token ring network: shared variables and stations

To the MAC protocol machine, the shared variables *inbuf* and *outbuf* model the physical layer and transmission medium. The variable is an array of symbols; the length or number of elements in the array is equal to more than the maximum message length. The array elements are also allowed to be empty.

Specification of the network stations

The station specification, or MAC protocol machine, consists of the finite state machine given in Figure 3; the local variables in Table 2; the predicate action in Table 3; and the two timers, TRT and THT, shown in Figures 4, 5 and Tables 5 and 6. Table 4 contains a listing of the transition names and other important acronyms, with a brief explanation of their meaning.

There are 20 states in the MAC protocol machine; the initial state is zero. The station remains in states 0 through 7 as long as it has no data frames to transmit to other stations. It passes to state 10 from state 0 when it receives a data frame to send; that is, when a protocol data unit (PDU) becomes queued for transmission.

From state 0, three other transitions in addition to the *PDU-Q* are possible; these involve the receiving of messages. These three receiving transitions are *token*, *rcv-F*, and *pass-F*. The first merely involves the receipt of the token, and (since there are no messages to transmit) passing it on to the next station. The second and third both receive an incoming data frame. In *rcv-F*, the data frame is addressed to the station itself, and so must be copied, symbol by symbol, at the same time it is repeated

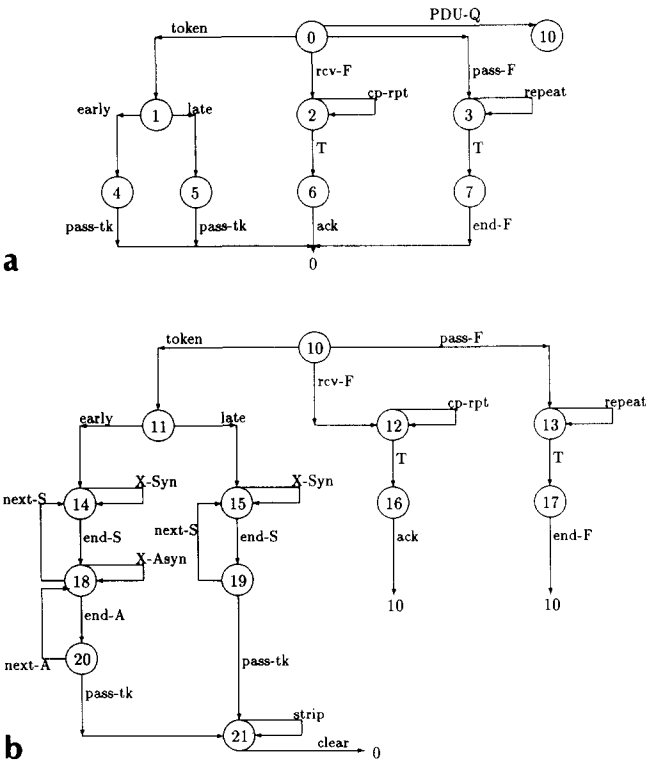

Figure 3. State diagram for the FDDI token ring protocol. (a) No data to transmit; (b) data to transmit

Table 2. Local and timer variable specification

Variable name	*Range*	*Initial value*	*Purpose*
Late-cnt	0 . . inf	0	counter for TRT timer
T-opr	integer		TTRT value (constant)
A-buf	array [1 . . n] of buffer	Ø	asynchronous messages to be sent
S-buf	array [1 . . n] of buffer	Ø	synchronous messages to be sent
msg-buf	*buffer*	Ø	store incoming messages
i	1 . . *MFL* + 1	1	pointer into A-buf
j	1 . . *MFL* + 1	1	pointer into S-buf
in	1 . . *MFL* + 1	1	pointer into in-buf
out	1 . . *MFL* + 1	1	pointer into out-buf
F-cnt	0 . . inf	0	frame counter (all frames)
S-cnt	0 . . inf	0	synchronous frame counter
err	boolean (T, F)	*F*	set when error detected
max	0 . . inf		limit to synchronous transmission
TRT-val	0 . . inf	*T-opr*	counter for the TRT timer
THT-val	0 . . inf	0	counter for the THT timer
ENABLED	boolean (T, F)	*F*	used to start the THT timer

Table 3. Predicate-action table for FDDI MAC machine (UB is time limit)

Transition	*UB*	*Enabling predicate*	*Action*
PDU-Q	1	$Abuf0(i) \neq \emptyset \vee S\text{-}buf(j) \neq \emptyset$	
token(=L.B.)	7	$inbuf[1 . . 7] = (I, J, K, 0, 0, T, T)$	$inbuf \leftarrow \emptyset$; $S\text{-}cnt \leftarrow 0$
early	1	$Late\text{-}cnt = 0$	$THT\text{-}val \leftarrow TRT\text{-}val$; $TRT\text{-}val \leftarrow T\text{-}Opr$
late	1	$Late\text{-}cnt > 0$	$Late\text{-}cnt \leftarrow 0$
pass-tk(=L.B.)	7	TRUE	$outbuf[1 . . 7] \leftarrow (I, J, K, 0, 0, T, T)$
rev-F	1	$inbuf[5] \in \{1, 2\} \wedge inbuf[6 . . 7] = MA$	$in \leftarrow 1$
cp-rpt	1	$inbuf[in] \neq T$	$msg\text{-}buf[in], outbuf[in] \leftarrow inbuf[in]$; $in \leftarrow in + 1$
T	1	$inbuf[in] = T$	$outbuf[in] \leftarrow T$; $inbuf \leftarrow \emptyset$; $in \leftarrow in + 1$
end-F	3	TRUE	$outbuf[in, in + 1, in + 2] \leftarrow (err, inbuf[in + 1, in + 2])$
ack	3	TRUE	$outbuf[in, in + 1, in + 2] \leftarrow (err, 1, 1)$
pass-F	1	$inbuf[5] \in \{1, 2\} \wedge inbuf[6 . . 7] \neq MA$	$in \leftarrow 1$
repeat	1	$inbuf[in] \neq T$	$outbuf[in] \leftarrow inbuf[in] \leftarrow in + 1$
X-Syn	1	$S\text{-}buf[j, out] \neq \emptyset$	$outbuf[out] \leftarrow S\text{-}buf[j, out]$; $out \leftarrow out + 1$
X-Asyn	1	$A\text{-}buf[i, out] \neq \emptyset \wedge (S\text{-}cnt = max \vee S\text{-}buf[j] = \emptyset)$	$outbuf[out] \leftarrow A\text{-}buf[i, out]$; $out \leftarrow out + 1$
end-S	3	$S\text{-}buf[j, out] = \emptyset$	$outbuf[out, out + 1, out + 2] \leftarrow (T, 0, 0)$; $S\text{-}cnt, F\text{-}cnt \leftarrow S\text{-}cnt + 1$; $j, out \leftarrow j \oplus 1, 1$
end-A	3	$A\text{-}buf[i, out] = \emptyset$	$outbuf[out, out + 1, out + 2] \leftarrow (T, 0, 0)$; $F\text{-}cnt \leftarrow F\text{-}cnt + 1$; $i, out \leftarrow i \oplus 1, 1$
next-S	1	$S\text{-}cnt < max \wedge S\text{-}buf[j] \neq \emptyset$	
next-A	1	$THT\text{-}val > 0 \wedge A\text{-}buf[i] \neq \emptyset$	
strip	MFL	$inbuf[6 . . 7] = MA \wedge F\text{-}cnt > 0$	$inbuf \leftarrow \emptyset$; $F\text{-}cnt \leftarrow F\text{-}cnt - 1$
clear	1	$F\text{-}cnt = 0$	
TRT-watch	0	$TRT\text{-}val = 0$	$TRT\text{-}val \leftarrow T\text{-}opr$; $Late\text{-}nct \leftarrow Late\text{-}cnt + 1$
CRASH	1	$Late\text{-}cnt > 1$	notify SMT, terminate ring operation

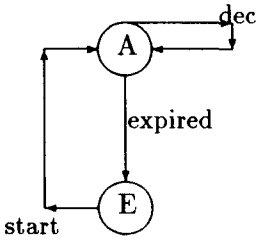

Figure 4. State diagram and variable for the TRT *timer*

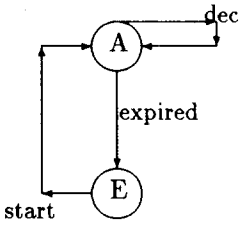

Figure 5. State diagram and variable of the THT *timer*

to the downstream station. When the end of the frame is reached, indicated by the symbol 'T' (the ending delimiter), the transition to state 6 is made, and the acknowledgment is sent by setting the bits in the frame status, immediately following the 'T' symbol.

Table 4. Meanings of transition names and acronyms

Name	*Meaning*
PDU-Q	a frame (PDU) is queued for transmission
PDU	Protocol Data Unit, a frame or message
token	receiving incoming token
rcv-F	accept frame, DA = MA
pass-F	pass frame on to next station
early	token arrived before TTRT
late	token arrived after TTRT
TTRT	Target Token Rotation Time
cp-rpt	copy and repeat symbol to next station
repeat	repeat symbol to next station
T	ending delimiter for frame or token
ack	acknowledgment of frame
end-F	send end of frame
X-Syn	transmit synchronous frame
X-Asyn	transmit asynchronous frame
MA	my address
DA	destination address
SA	source address
next-S	begin sending the next Sync. frame
next-A	begin sending the next Async. frame
end-S	end of transmission, Sync. frame
end-A	end of transmission, Async. frame
pass-tk	pass the token to next station
strip	strip my frames from the ring
clear	ring is clear of my frames
SMT	station management

The *pass-F* transition passes the frame, symbol by symbol, to the next station without copying it. When the end of frame is reached, the 'T' transition takes the machine to state 7, and the frame status field is repeated without setting the acknowledgment bits.

In state 10 a message (PDU) is ready for transmission, so that when the token arrives the station will 'claim it' (not pass it on) and transmit. The *rcvF* and *pass-F* transitions take the same actions as from state 0.

When the station claims the token, it will pass to state 11. There are two cases; either the token is 'early' or 'late'. If the token is early – indicated by *Late-cnt* = 0 – then the 'early' transition will be taken to state 14. This means that the token has circulated the ring in less than one TTRT, leaving some time for the transmission of asynchronous frames as well as synchronous frames.

In state 14, one synchronous frame is transmitted, symbol by symbol (see the *X-Syn* transition). At the end of the frame, the machine moves to state 18; from state 18, the machine will return to 14 for the next synchronous frame transmission, until all are sent (*S-Buf* [*j*] is empty), or until the maximum number allowed is reached (*S-cnt* reaches *max*). Then, in state 18, one asynchronous frame will be sent, symbol to symbol, at the end of which the transition to state 20 is made. From state 20 the machine will return to 18 for the next asynchronous frame if one is queued and time permits (see the *next-A* transition).

The symbol '⊕' is used to increment the pointers *i* and *j* in the *next-A* and *next-S* transitions. It is essentially modulo addition, resetting the pointer to the first buffer element after the last is reached.

From state 20, when no more asynchronous frames are queued *or* when the timer THT expires, the token will be passed as the machine transitions to state 21. The station remains in this state in order to strip its own frames off the ring, as they return to it from the opposite shared buffer.

The time limit on the *pass-tk* transition, as well as the *token* transition, is specified as a lower bound as well as an upper bound. This is because the timing of these is determined precisely by the physical bit rate of the hardware. This also serves to make the choice between transitions in states 19 and 20 deterministic; if the *next-A*/*next-S* transitions are enabled in these states, then they will occur before the token is passed.

The specification assumes that the ring is sufficiently long that the transmitting station will reach the strip state, state 21, before its frames return. If this were not the case, then this stripping part can be specified as a separate parallel machine.

Consider next the case that the token is late; this means that the TRT timer has expired once, that is *Late-cnt* > 0 (see the *late* transition). Then the station may only transmit synchronous frames, taking the transition to state 15.

In state 15 and 19, the station transmits its synchronous frames exactly as it is done in states 14 and 18. When no more synchronous frames are queued, or when the maximum number of these have been sent, the token is passed, taking the machine to state 21, where the frames which were transmitted are removed from the ring.

When all frames which the station transmitted have

been removed, indicated by the frame counter variable *F-cnt*, the station returns to the initial state.

Each of the two timers, TRT and THT, is specified as a simple two state machine, and a variable which serves as a counter. In the *active* state, state A, the timer repeatedly decrements the value in the variable. When this value reaches zero, the timer passes to the *expired* state, state E. The upper and lower bounds of this transition are both set to one time unit, so that the machine is forced to act as a timer.

The basic unit of time which was chosen was the time to transmit one symbol, as this is the unit which the MAC layer passes to the physical layer. For each symbol transmitted, the physical layer transmits 5 bits. Four of these bits are data bits; the 100 Mbit/s refers to the data bits. The rate of transmission of *all* bits is 125 Mbit/s (125 Mbaud is the baud rate, the maximum number of times the signal can change per second). Thus, at 100 Mbit/s, the basic unit of time is 0.04 microseconds. That is, each 0.04 μs, 1 symbol with 4 data bits and 1 overhead bit is transmitted.

The timer variables are shared with the protocol machine of the station. These are used by the station to reset the timer when desired and to start it running. The variable *ENABLED* is also used by the station to start the THT timer running, when needed. The exact procedure is specified in the predicate-action table in Table 5.

There are two transitions specified in the MAC machine table (see Table 2) which are not shown in the state diagram; this is because these transitions can be taken from *any* state. The *TRT-watch* transition becomes enabled whenever the TRT timer expires. This transition immediately resets the timer, and increments variable *Late-cnt*. By setting the upper bound on time of this transition to 0, we insure that the timer is reset immediately. This is a way of modelling an interrupt mechanism.

The second transition not shown is called *CRASH*; this is the termination of the ring operation, if the token fails to circulate within twice the TTRT. If the protocol operates as specified, this transition should never be taken unless forced by some external event, such as a hardware failure.

Initial state of the network

The initial system state is specified as follows:

(1) each protocol machine is in state 0;
(2) each shared variable between protocol machines is empty, *except* exactly one, which contains the token;
(3) The TRT timer is initially in state 'A', the active state, with the variable *TRT-val* set to *T-opr*;
(4) The THT timer is initially in state 'E', the expired state;
(5) the intital values of the local variables are as specified in Table 1.

Startup requirements. The following restrictions apply upon the startup of the ring. These could have been included in the formal specification; however, we prefer to simply state them here, rather than further complicating the specification:

1 The TRT timer of each station is started for the first time, *when the station first receives the token.*
2 The first time a station receives the token, no frames are transmitted; the token is passed on to the next station.
3 The second time the token is received, no *asynchronous* frames are transmitted.
4 The sum of the synchronous allotments for all stations is at most one frame time *less* than the target token rotation time (TTRT).

ANALYSIS FOR SAFETY AND LIVENESS PROPERTIES

Previous work[4, 5] proved that the timing requirements of FDDI are satisfied; however, that work did *not* give a formal description of the protocol, but *assumed* that the network nodes performed the protocol correctly. The proofs were based on the assumption that the nodes perform the protocol correctly; that there are no errors such as deadlocks, and that the functions of token passing and data transfer are carried out correctly.

Table 5. Predicate-action table for the TRT timer

Name	*Time*	*Enabling predicate*	*Action*
dec	1	*TRT-val* > 0 ∧ *operational*	*TRT-val* ← *TRT-val* − 1
expired	1	*TRT-val* = 0	*Late-cnt* ← *Late-cnt* + 1
start	0	*TRT-val* > 0	

Table 6. Predicate-action table for the THT timer

Name	*Time*	*Enabling predicate*	*Action*
dec	1	*THT-val* > 0	THT-val ← *THT-val* − 1
expired	1	*THT-val* = 0	ENABLED ← *false*
start	0	ENABLED = *true*	

The results of this paper are complementary: we give a formal description of the protocol and prove that a node operating according to this description does so without deadlocks, and that the progress functions of token and frame passing are indeed executed. These are necessary conditions for the timing requirements to be satisfied.

Safety properties are a guarantee that something bad does *not* happen, while a liveness property is a guarantee that something good *does* happen.

In Lemmas 1–3 we prove some basic characteristics of the protocol. These will simplify the proof of the theorem. A basic assumption in the following results of this section is that the ring is not forced to reinitialize, or 'crash', due to failure to rotate within the time limits; in other words, the *CRASH* transition is not taken. The validity of this assumption is confirmed in the next section.

The reader may verify the intermediate proof steps by examining the state diagram and corresponding predicate-action table.

Lemma 1 *If the token is in the input buffer* inbuf *for station* i, *and station* i *is in state 0, then within a finite number of time units the token will be removed from* inbuf *with station* i *in state 0.*

Proof The conditions above enable the *token* transition to state 1. From state 1, either the *early* or *late* must be taken to states 4 or 5. From these, *pass-token* writes the token into *outbuf*, returning to state 0. Each of these transitions has a finite upper time bound, so the sequence is bounded. □

From Lemma 1, it follows that by observing the sequence of transitions taken and the upper bound on each, it may easily be verified that the token will be passed within *15 times units.*

Lemma 2 *If a data frame is in the input buffer* inbuf *for station* i, *and station* i *is in state 0 (10), then within a finite numbe. of time units the frame will be removed from* inbuf *and appear in* outbuf, *with the station again in state 0 (10).*

Proof Suppose station *i* is in state 0. If the DA part of *inbuf* contains *I*'s address, then the *rcv-F* transition to state 2 will be taken; otherwise the *pass-F* transition to state 3 is taken. In either of these, the frame will be copied symbol by symbol into *outbuf*. When the end of frame is reached, as indicated by the ending delimiter *T*, the node transitions to states 6 or 7, and then returns to state 0. As each transition is time bounded, so is the entire sequence.

The proof beginning in state 10 is identical. □

Lemma 1 states that when a station receives the token, and has no data to transmit, that it will pass the token on to the next station. Lemma 2 states that a station also passes a data frame. Note that Lemma 2 does not apply to a station's own frames; the station must strip its own frames off the ring. But this is taken care of by specifying the state; a station will be in state 21 when receiving its own data frames. Also note that in both Lemmas, the station *always* returns to either state 0 or state 10 after processing its incoming data. This is a key in proving freedom from deadlocks.

Lemma 3 is an extension of Lemma 1, to include the case in which a station gets the token and transmits.

Lemma 3 *If the token is in the input* inbuf *for station ,, and station* i *is in state 10, then within a finite number of time units, the token will be removed from buffer* inbuf *and appear in buffer* outbuf, *and the station will return to the initial state 0.*

Theorem 1 (Safety) *Excluding the* CRASH *transition, the FDDI protocol as specified is free from deadlocks.*

Proof Assume that the stations are ordered 1, 2, . . , *n*, and that initially the token in the input buffer of station 1.

It is sufficient to show that the token continues to circulate indefinitely. From the initial system state, each machine is in state 0; any machine receiving a PDU to transmit moves to state 10. Thus each machine is in either state 0 or state 10 when the token first appears in its input buffer.

By Lemmas 1 and 3, station 1 will receive the token, pass it on to station 2, and return to state 0 or 10. By Lemma 2, any data frames which station 1 transmits will be passed by each station to the next; then each station will return to state 0 or 10. Station 1 will receive the returning data frames, strip them, and return to state 0.

Upon receiving the token from station 1, station 2 will go through the same actions, eventually passing the token to station 3, then station 4, etc., until the token eventually returns to station 1. □

Corollary 1 *Excluding the* CRASH *transition, the FDDI protocol as specified is free from nonexecutable transitions.*

This corollary states that every transition is executable – that is, there are no unreachable 'lines of code' in the program – with the possible exception of the *CRASH* transition, which of course we hope will not be executed. The corollary may be proved by working through the proofs of the Lemmas and theorem above, and noting that all transitions are executable at some point.

Theorem 2 (Liveness) *If station* i *transmits a data frame to station* j, i ≠ j, *then the frame will appear in the local* msg-buf *of station* j *within a finite number of time units.*

Proof Suppose that the stations downstream from *i* are $l_1, l_2, \ldots, l_k, j$ in order. Station *i* transmits the frame by placing it, symbol by symbol, into *outbuf*. Next l_1 will remove the frame and pass it to l_2, by Lemma 2. This continues until l_k places it into the input buffer if station *j*.

Station *j* is in either state 0 or 10 (because station *i* has the token). Since the frame is addressed to *j*, the *DA* field contains *j*'s address; this enables the *rcv-F* transition. Now in state 2 or 12, *j* will copy the message into *msg-buf*. When the end of the message is reached the entire message will be in this buffer. □

CONCLUSIONS

A formal specification and analysis of the FDDI token ring protocol has been presented using a model called *systems of communicating machines.* The specification included a specification of three machines: the main station or network note, and two timers. The physical layer of the network was modelled by a set of variables shared by adjacent nodes.

Analysis showed that the specification was free from deadlocks, and that the protocol possesses the critical *liveness* or progress property.

The emphasis is on the MAC level of the network. Each station is modelled as a finite state machine, having 20 states, together with a set of local variables and two timers. The local variables include buffers to store input and output messages and pointers. The timers are a separate machine which ticks at constant rate, decrementing a counter. The station is able to use and control the timers through variables which it shares with them. The unit of time used corresponds to the time needed to transmit one symbol; in this case it is 0.04 microseconds.

The paper also extends the model *systems of communicating machines* to include the explicit timing of transitions. Previous work had modelled time as a sequence of events, not assigning specific time limits to transitions.

REFERENCES

1 **Lundy, G M** *Systems of Communicating Machines: A Model for Communicating Protocols*, PhD Thesis, School of Information and Computer Science, Georgia Institute of Technology (July 1988)

2 **Lundy, G M and Miller, R E** 'Analyzing a CSMA/CD protocol through a systems of communicating machine specification', *IEEE Trans. Commun.* (to appear)

3 *Protocol Specification, Testing and Verification, Volumes I–XI*, North-Holland, Netherlands (1981–1991)

4 **Johnson, M J** 'Proof that timing requirements of the FDDI token ring protocol are satisfied', *IEEE Trans. Commun.*, Vol 35 No 6 (June 1987) pp 620–625

5 **Sevcik, K C and Johnson, M J** 'Cycle time properties of the FDDI token rink protocol', *IEEE Trans. Softw. Engl.*, Vol 13 No 3 (1987) pp 376–385

6 **Akyildiz, I F, Chiola, G, Kofman, D and Korezlioglu, H** 'Stochastic Petri net modeling of the FDDI network protocol', *Protocol Specification, Testing and Verification XI*, North-Holland, Netherlands (1991)

7 *FDDI Token Ring Media Access Control (MAC-2)*, ANSI Standard X3.139–1987, REV-10, ANSI, USA (1987)

8 *FDDI Media Access Control (MAC-2)*, Rev 4.0, ANSI, USA (1990)

9 **Grow, R M** 'A timed token protocol for local area networks', *Electro '82* (May 1982)

10 Institute of Electrical and Electronics Engineers,*IEEE Standard 802.5, Token Ring Access Method and Physical Layer Specifications*, IEEE, New York (1985)

11 **Ross, F E** 'An overview of FDDI: the fiber distributed data interface', *J. Selected Areas in Commun.*, Vol 7 No 7 (September 1989) pp 1043–1051

12 **Ulm, J N** 'A timed token ring local area network and its performance characteristics', *Proc. 7th Conf. on Local Computer Networks*, IEEE Press, New York (February 1982)