MODELING PHYSICAL LAYER PROTOCOLS USING COMMUNICATING FINITE STATE MACHINES

Mohamed G. Gouda and Khe-Sing The

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

Abstract

We illustrate the usefulness of communicating finite state machines in modeling a number of physical layer protocols that include (i) an asynchronous start-stop protocol and (ii) a protocol for synchronous transmission with modems. Each protocol is modeled as a network of four finite state machines that communicate by exchanging messages over unbounded, FIFO channels. (Two machines are used to model the protocol itself, while the other two are used to model its interface to the upper data link protocol in the protocol hierarchy.) We outline a methodology to verify communication boundedness and progress for each protocol model. The methodology is based on three techniques that were proposed earlier to verify networks of communicating finite state machines; they are network decomposition, machine equivalence, and closed covers.

1. Introduction

The physical layer is the lowest layer in a protocol hierarchy [1,7,15]; its function is to transmit frames of data characters or bits across a physical line between two adjacent computers in a computer network [1,7,14]. Most of the available literature on physical layer protocols concentrate on defining their electrical specifications and pin descriptions, and ignore their procedural characteristics and interface behaviours. Although the procedural characteristics of these protocols may seem simple and straightforward at first, the task of modeling them formally and proving their correctness can become quite complicated.

The objective of this paper is three-fold:

i. Provide formal models based on communicating finite state machines to two important and widely used physical layer protocols: (a) an asynchronous start-stop protocol, and (b) a protocol for synchronous transmission with modems. We believe that by going through these formal models, one can appreciate the function of these protocols and their different ways of achieving this function.

ii. Illustrate the usefulness of communicating finite state machines as a convenient formal model for defining communication protocols. (For more examples in using communicating finite state machines to model communication protocols we refer the reader to [2,3,4,5,6,16,17].)

iii. Demonstrate the utility of recent verification methodologies [8,10,11,12,13] for networks of communicating finite state machines in establishing the correctness of "real" and nontrivial protocol models.

Following the introduction, the paper is organized as follows. Networks of communicating finite state machines are briefly presented in Section 2. Then, in Section 3, we discuss how to model a physical layer protocol as a network of four communicating finite state machines: two machines model the protocol itself and the other two model the protocol's interface with the data link protocol. In Section 4, we use this model to formally define two examples of physical layer protocols. A verification methodology for these protocol models is outlined in Section 5. Finally, concluding remarks are in Section 6.

2. Networks of Communicating Finite State Machines

A *network* W is a tuple $[M_1,...,M_n]$, where $n \geq 2$, and for $i=1,...,n$, M_i is a finite state machine that communicates with other finite state machines in W by exchanging messages via one-directional, unbounded, FIFO channels. Next, we define communicating finite state machines.

A *communicating finite state machine* M_i in a network $W=[M_1,...,M_n]$ is a labeled directed graph with three types of edges, namely sending, receiving, and internal edges:

- a *sending edge* in M_i is labeled with $-g/M_j$,
- a *receiving edge* in M_i is labeled with $+g/M_j$, and
- an *internal edge* in M_i is labeled with the empty sequence E,

where g is a *message* from a finite set G of messages, and M_j is a machine, other than M_i, in W. One of the nodes in M_i is identified as its *initial node*, and each node in M_i is reachable by a directed path from the initial node. For convenience, we assume that M_i is "nonterminating," i.e., each node in M_i has at least one outgoing edge.

A *state* s *of a network* $[M_1,...,M_n]$ is an $n \times n$ matrix, where an entry $s_{i,j}$ in the i-th row and j-th column is defined as follows:

i. If $i=j$, then $s_{i,i}$ is a node in machine M_i.

ii. If $i \neq j$, then $s_{i,j}$ is a sequence of messages from the set G; in this case, $s_{i,j}$ is called the *content of the channel from* M_i *to* M_j *in* state s.

The *initial state* s^0 of a network $[M_1,...,M_n]$ is a state of the network whose entries $s^0_{i,j}$ are defined as follows:

i. If $i=j$, then $s^0_{i,i}$ is the initial node in M_i.

ii. If $i \neq j$, then $s^0_{i,j}$ is the empty sequence E.

Let s and s' be two states of a network $[M_1,...,M_n]$, and let e be an edge from node v to node v' in machine M_i. State s' is said to *follow* s *over* e iff one of the following three conditions is satisfied:

i. e is a sending edge labeled $-g/M_j$, and each entry in s is identical to its corresponding entry in s' except (possibly) for $s_{i,i}$ and $s'_{i,i}$, and for $s_{i,j}$ and $s'_{i,j}$ whose values are defined as follows:
$s_{i,i}=v$, $s'_{i,i}=v'$, and $s'_{i,j}=s_{i,j}.g$, where "." is the usual string concatenation operator.

ii. e is a receiving edge labeled $+g/M_j$, and each entry in s is identical to its corresponding entry in s' except (possibly) for $s_{i,i}$ and $s'_{i,i}$, and for $s_{j,i}$ and $s'_{j,i}$ whose values are defined as follows:
$s_{i,i}=v$, $s'_{i,i}=v'$, and $s_{j,i}=g.s'_{j,i}$.

iii. e is an internal edge labeled E, and each entry in s is identical to its corresponding entry in s' except (possibly) for $s_{i,i}$ and $s'_{i,i}$ whose values are defined as follows:
$s_{i,i}=v$ and $s'_{i,i}=v'$.

Let s and s' be two states of a network W. State s' is said to *follow* s iff there exists an edge e in one of the machines in W such that s' follows s over e.

Let s and s' be two states of a network W. s' is *reachable from* s iff either $s=s'$ or there exist a sequence $s^0,s^1,...,s^r$ of states of W such that $s^0=s$, $s^r=s'$, and for $i=0,...,r-1$, s^{i+1} follows s^i. A state of a network is *reachable* iff it is reachable from the initial state of the network.

The communication of a network $W=[M_1,...,M_n]$ is said to be *bounded by* K iff for every reachable state s of W, and for every $i=1,...,n$ and every $j=1,...,n$, if $i \neq j$ then $|s_{i,j}| \leq K$, where $|x|$ is the number of messages in sequence x. The communication is *bounded* iff it is bounded by some positive integer K.

A state s of a network W is called an *unspecified reception state for machine* M_i in W iff $s_{i,i}$ is a receiving node in machine M_i, and for every outgoing edge e of node $s_{i,i}$, if e is labeled with $+g/M_j$, then $s_{j,i}$ is a nonempty sequence of the form g'.x, where $g \neq g'$. The communication of W is said to be *free from unspecified receptions* iff no reachable state of W is an unspecified reception state (for some machine in W).

Let s be a state of a network W, and let M_i be a machine in W, and $\{M_a,...,M_b\}$ be a set of machines in W. M_i is said to be *blocked by* $\{M_a,...,M_b\}$ *at* state s iff the following condition holds for each outgoing edge e

of node $s_{i,i}$ in machine M_i. Edge e is labeled with some $+g/M_j$ such that

(i) M_j is in the set $\{M_a,...,M_b\}$, and

(ii) either $s_{j,i}=E$ (the empty sequence) or $s_{j,i}=g'.x$, where $g'\neq g$.

A set D of machines in W is called a *deadlock set at state* s iff each machine in D is blocked by a subset of D at s. A state s of a network W is called a *deadlock state* iff there exists a set of machines in W that constitutes a deadlock set at state s. The communication of W is said to be *free from communication deadlocks* iff no reachable state of W is a deadlock state.

The communication of a network W is guaranteed to *progress indefinitely* iff the communication of W is free from unspecified receptions and communication deadlocks. The communications of the two protocol models presented in this paper can be shown to progress indefinitely; an outline of the verification methodology is given in Section 5.

3. Modeling Physical Layer Protocols

The physical layer is the lowest layer in a protocol hierarchy; it serves as a communication medium to the data link layer [1,7,15]. As shown in Figure 1, the data frames sent by the data link layer at one site are first delivered to the physical layer at that site, then transmitted as a stream of characters over the communication line between sites, and finally assembled into frames before delivery to the data link layer at the other site. This arrangement can be modeled as a network of four communicating finite state machines [P,M,N,Q], where M and N model the physical layer protocol, and P and Q model the interface of the data link layer to the physical layer (see Figure 2).

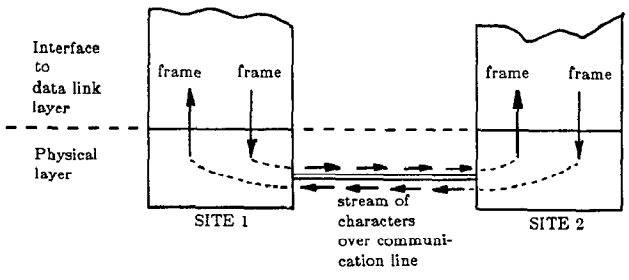

Figure 1: Physical layer.

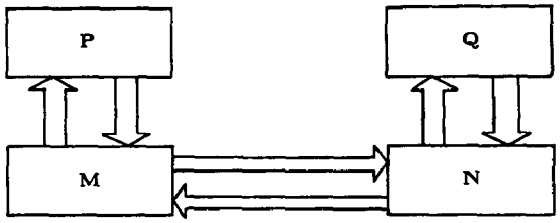

Figure 2: Modeling a physical layer protocol as a network of four machines.

In Section 5, we model various physical layer protocols as networks [P,M,N,Q] that differ in the way the communication between M and N is governed, but provide a uniform interface through P and Q. This means that the interface machines P and Q are the same for all the protocol models discussed in Section 5, but M and N change from one protocol model to another. The interface machines P and Q are defined in this section, and the different versions of machines M and N are discussed in Section 5.

The chosen interface between the data link layer and the physical layer satisfies the following requirements:

i. *Basic full duplex operation:* The interface allows the data link layer at each site to send frames to the other site, and to receive frames from the other site, possibly at the same time.

ii. *Simple operation:* For the data link layer at one site to send a frame, it simply sends the frame to the physical layer at its site. For it to receive a frame, it simply sends a "next" message to the physical layer at its site, then waits for the frame.

iii. *Friendly operation:* Before the data link layer at one site can send one frame to the other site, it must also be ready to receive one frame from the other site.

iv. *No error detection:* The interface does not guarantee that all sent frames will be delivered correctly. It is the function of the data link layer, not the physical layer, to detect frame corruption and loss [1,14,15].

Based on these requirements, the communicating finite state machine P can be defined as shown in Figure 3. (Machine Q is identical to P except that it communicates with N instead of M.) Machine P exchanges the following messages with machine M.

- "frame" denotes a data frame,
- "next" denotes a request from P to M to send the next data frame, and
- "noframe" denotes a reply of "no frame is available" from M to P.

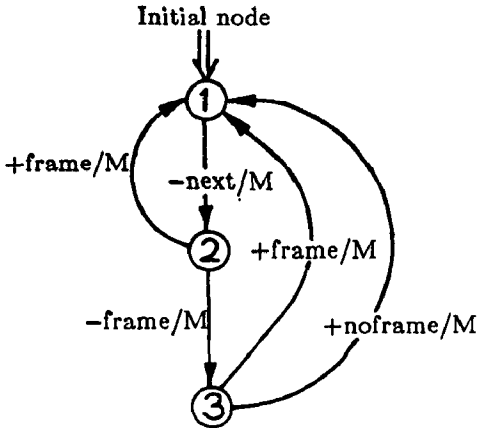

Figure 3: Machine P.

Starting from node 1, P can only send a request "next" to M. It then either waits to receive the requested frame from M and returns to node 1, or sends a data frame to M. As discussed later, M is expected to forward this data frame to N. If in the mean time M receives a frame from N, it sends it to P as a reply to its request; otherwise it sends a "noframe" reply to P. In either case P returns to its initial node.

4. Examples of Physical Layer Protocols

In this section, we discuss formal models of two physical layer protocols: an asynchronous start-stop protocol, and a protocol for synchronous transmission with modems. For convenience, we adopt the following two abbreviations in drawing the communicating machines in this section:

1. An edge label of the form $-g/M_i$ ($+g/M_i$) can be written as -g (+g), when M_i can be implicitly understood from g. For example, the edge label -frame/P in machine M will be written as -frame, because M sends "frame" only to P.

2. Two or more edges from node v to node v' can be drawn as one edge from v to v' with multiple labels.

4.1. An Asynchronous Start-Stop Protocol

This protocol is used to transmit frames of data characters over a single full duplex communication line [14]. A data frame consists of an integral number of contiguous data characters, where a character has a fixed length of bits. The bits are transmitted serially, hence, there should be an agreement between the sender and receiver on the duration of a bit over the communication line, so the state of the line can be properly sampled.

The protocol provides a mechanism to delineate frames and characters. A data character consists of a START bit (usually a "0" bit), fixed number of data bits, and a STOP bit (usually a "1" bit). When there are no data characters to transmit over the line, idle characters are transmitted instead. An idle character consists of all idle bits (usually "1" bits). Two successive data frames should be separated by at least one idle character. The receiver can detect the beginning of a frame by sensing the START bit of the first data character in the frame, and the end of a frame by the idle character following the last data character.

Formal modeling: Figure 4 shows machine M. (Machine N is identical.) M and N exchange the following messages:

- "idle" denotes an idle character,
- "start" denotes the START bit of a data character,
- "data" denotes the sequence of data bits in a data character, and
- "stop" denotes the STOP bit of a data character.

The initial node, node 1, of M is the "not ready" node, since at this node M has not yet received a "next" from P, and so it cannot accept any data characters from N. If at node 1 M receives a "next" message from P, it proceeds to node 2 and becomes ready to communicate with N. On the other hand, if at node 1 M receives a data character from N, it proceeds to node 20, the "error" node, and discards the incoming data characters as indicated by the dashed receiving edges. M proceeds from node 20 to node 2 when it receives the first "idle" message from N after a "next" message from P.

Node 2 is the "ready" node, where M keeps sending and receiving "idle" messages until it receives a frame from P (then goes to node 3), or a data character from N (then goes to node 4).

Nodes 3 and 16 are "data sending" nodes, where M sends data characters to N, and receives idle characters from N. If at node 3 M finishes sending the characters of the current frame, it recognizes that it has received no data characters from N, so it sends a "noframe" message to P and returns to the initial node. On the other hand, if at node 3 M receives a data character from N, it proceeds to node 8.

Node 4 is the "data receiving" node, where M receives data characters from N, and sends idle characters to N. If at node 4 M receives an "idle" indicating the end of the received frame, it immediately delivers this frame to P. On the other hand, if at node 4 M receives a data frame from P, it proceeds to node 8.

Node 8 is the "data sending and receiving" node, where M and N exchange data characters. From this node M can proceed to node 4 (16) whenever it sends (receives) an "idle" message. Notice that when M receives an "idle" message indicating the end of the current received frame at node 15, it sends the received frame to P.

4.2. A Protocol for Synchronous Transmission with Modems

This protocol is used to transmit frames of data characters over a full duplex communication line with a pair of modems, one at each end [14]. The data bits are modulated in carrier signals, and full duplex communication can be achieved by selecting a different carrier frequency for each direction. The clocking signal is encoded in the modulating process and decoded from the sidebands of the carrier signal.

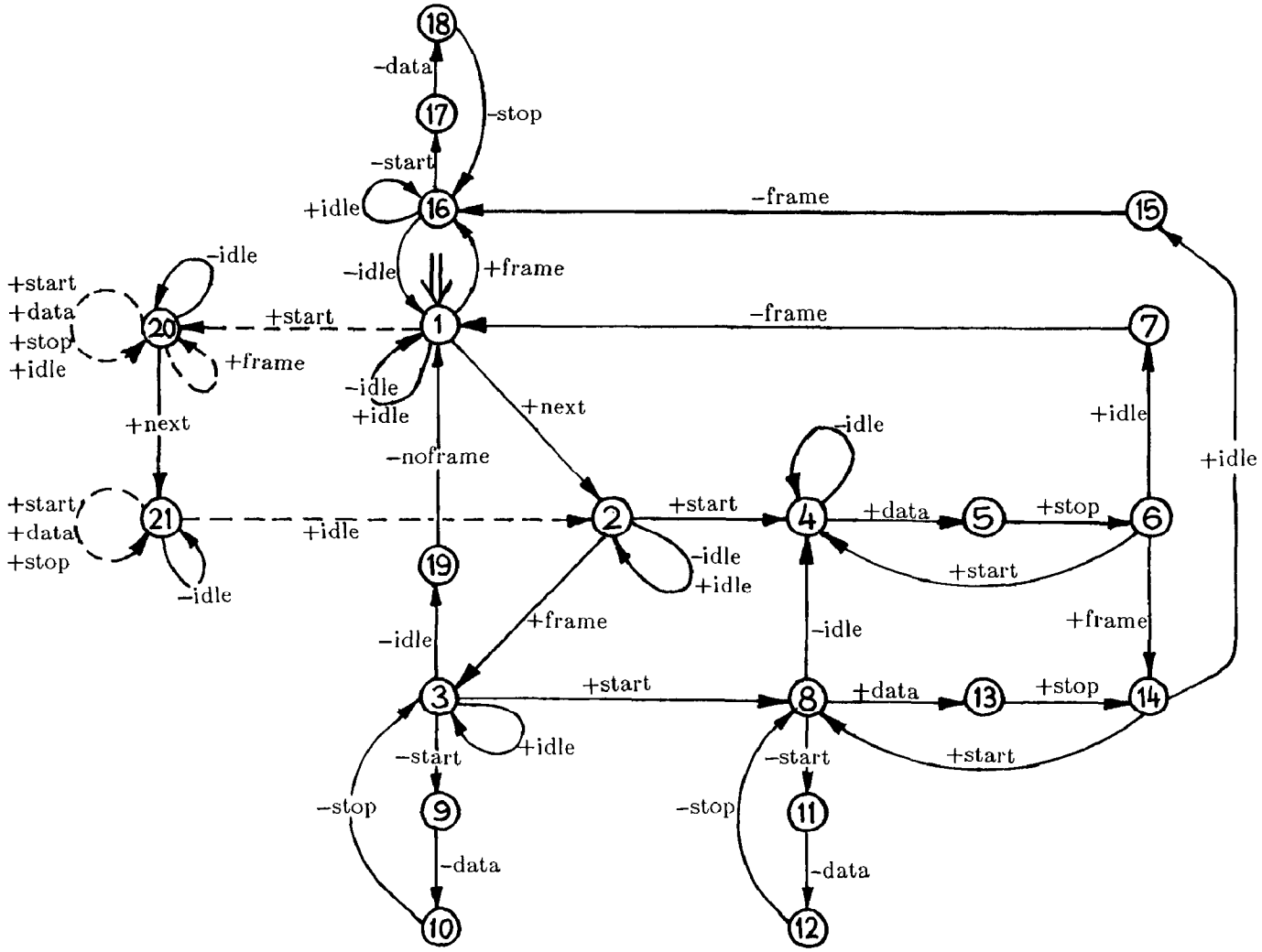

Figure 4: Machine M of the asynchronous start-stop protocol.

The existence of a clocking signal in this protocol necessitates a special synchronizing character (denoted by "sync") to be sent whenever either site has no data characters to send. A data frame consists of integral number of contiguous data characters and can be preceeded or followed by "sync" characters.

Unlike the previous protocol, here a connection should be established between the two sites before either site can send a data frame. A connection is indicated by the presence of carriers in both directions. During the interval from establishing to clearing a connection, at most one frame can be sent in each direction. After clearing a connection, new connections can be established to send more frames.

To establish a new connection, a site keeps on sending connect requests until it receives the incoming carrier, it then propagates its own carrier and the connection is established. Either site can request to clear the connection simply by ceasing to propagate its carrier, and the connection will be cleared after the incoming carrier has been turned off.

Formal modeling: The communicating finite state M (or N, which looks identical to M) of this protocol model is shown in Figure 5. The interpretation of messages exchanged between M and N are as follows:

- "conn" denotes a connect request.
- "car" is a message that represents the existence of carrier signals. ("-car" should be interpreted as starting to propagate the carrier, and "+car" should be interpreted as sensing an incoming carrier.)
- "nocar" is a message that represents the lack of carrier signals.
- "data" denotes a data character.
- "sync" denotes a "sync" character.

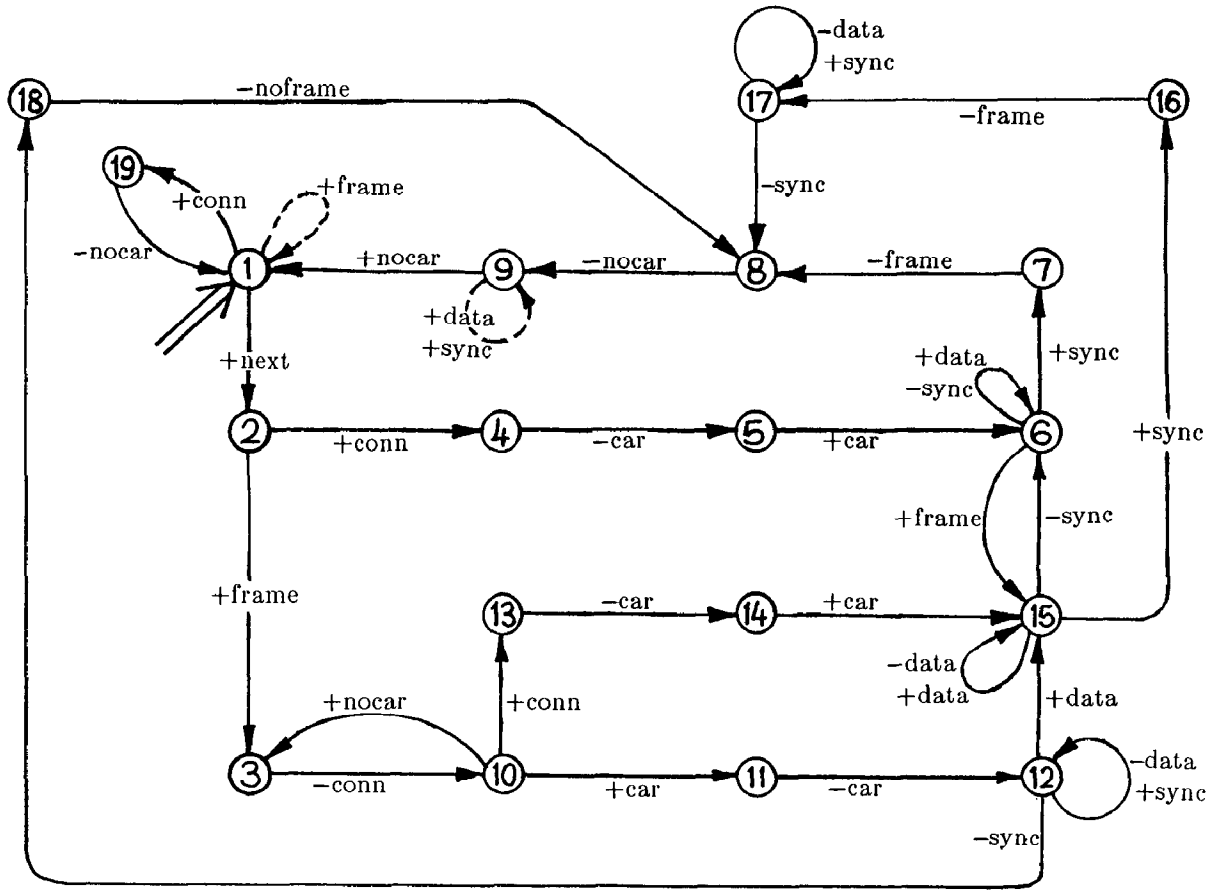

Figure 5: Machine M of the synchronous transmission protocol.

At node 1, the initial node, M has not yet received a "next" from P, and so it will reject any "conn" request from N. If M receives a "next" at node 1, it proceeds to node 2 where it becomes ready to accept a connect request and later starts receiving data characters at node 6, or to accept a frame from P and then tries to establish a connection (node 3).

Nodes 12 and 17 are "data sending" nodes, where M sends data characters to N, and receives "sync" characters from N. If M finishes sending the current frame while it is at node 12, it recognizes that it has received no data characters from N, so it sends a "noframe" message to P and prepares to clear the connection.

Node 6 is the "data receiving" node, where M receives data characters from N, and sends "sync" characters to N. M will come to this node if P does not have a frame to send, or if M has finished sending the frame from P to N but still has to receive data characters from N.

Node 15 is the "data sending and receiving" node, where M and N exchange data characters. M proceeds to node 15 when both M and N tries to establish a connection around the same time (node 13), or when M receives a data character from N at node 12, or when it receives a data frame from P at node 16.

When M receives a "sync" from N indicating the end of the current frame, it delivers the frame to P at either node 7 or 16.

M decides to clear the connection at node 8 by turning its carrier off, then ignores anything coming from N (as represented by the dashed self loop at node 9) until another connection is established. A frame that comes from P after M decides to clear the connection will also be lost, as represented by the dashed self loop at node 1. When both M and N have stopped propagating their carriers, then they are back at the initial nodes and the connection is cleared.

5. Verification of Physical Layer Protocol Models

As mentioned in Section 3, we model each physical layer protocol as a network [P,M,N,Q]. Proving indefinite communication progress of these networks is hard because (i) the two machines M and N are usually complex, and (ii) their communication is usually unbounded. To carry such proofs, we use three techniques: network decomposition, machine equivalence, and closed covers. These techniques are summarized next, but interested readers can find more details and examples about these techniques in [8], [10], and [11].

5.1. Network Decomposition

To prove that the communication of network [P,M,N,Q] in Figure 2 will progress indefinitely, we follow the next procedure.

1. Decompose the network into three networks, with two communicating machines each, [P,M'], [M",N"], and [N',Q], where

   M' (N') is identical to M (N) except that each sending edge to N (M) and each receiving edge from N (M) is replaced by an internal edge, and

   M" (N") is identical to M (N) except that each sending edge to P (Q) and each receiving edge from P (Q) is replaced by an internal edge.

2. Prove that network [P,M'] satisfies the following two conditions:

   a. The communication of the network will progress indefinitely, or equivalently every reachable state of the network is neither an unspecified reception state nor a deadlock state.

   b. For every reachable state s where machine M' is at a node v with an outgoing receiving edge, there exist two states s' and s" such that (i) s' is reachable from s over some edges in P, and (ii) s" follows s' over an outgoing receiving edge of node v.

3. Prove that network [N',Q] satisfies the same two conditions a and b in 2. (Replace M' by N' in condition b.)

4. Prove that the communication of network [M",N"] will progress indefinitely.

5. Proving 2, 3, and 4 guarantees that the communication of network [P,M,N,Q] will progress indefinitely. (The correctness of this assertion follows from the discussion in [10].)

Notice that the function of network decomposition in this procedure is to reduce the problem of proving some property for a four-machine network into proving some related properties for three

two-machine networks; it is a *divide and conquer* technique.

The proof needed in Step 2 of the above procedure is straightforward. This is because the communication of network [P,M'] is bounded; hence all reachable states of the network can be generated and examined to verify that conditions a and b are satisfied. Similarly, the proof needed in Step 3 is straightforward, since the communication of [N',Q] is also bounded.

Unfortunately, the proof needed in Step 4 is not straightforward since the communication of [M",N"] is usually unbounded. Therefore, to prove indefinite progress for the communication of network [M",N"], we resort to the technique of closed covers [8]. But before we use this technique, it is convenient to remove the internal edges from M" and N"; this requires a third technique known as machine equivalence.

5.2. Machine Equivalence

Let W be a network of communicating finite state machines, and let M be a machine that have some internal edges in W. Assume that each internal edge in M is in some directed cycle. Then, the equivalence-preserving transformations defined in [11] can be applied to transform M into an "equivalent" machine EM that has no internal edges. The equivalence is such that the communication of network W is guaranteed to progress indefinitely iff the communication of network EW is guaranteed to progress indefinitely, where EW is the same as W except that M is replaced by EM.

This result can be used to remove the internal edges from the two machines in [M",N"]. In particular, these equivalence-preserving transformations can be applied to the two machines M" and N" yielding the equivalent machines EM" and EN", respectively, such that (i) EM" and EN" have no internal edges, and (ii) the communication of [M",N"] is guaranteed to progress indefinitely iff the communication of [EM",EN"] is guaranteed to progress indefinitely. It remains now to discuss how to prove that the communication of [EM",EN"] will progress indefinitely. For that purpose, we need the technique of closed covers.

5.3. Closed Covers

As discussed in [8], to prove that the communication of a network will progress indefinitely, it is sufficient to exhibit a closed cover of the network. A *closed cover* C of network [EM",EN"] is a finite set of state schemas

$$\begin{bmatrix} v_1 & Y_1 \\ X_1 & w_1 \end{bmatrix}, \ldots, \begin{bmatrix} v_r & Y_r \\ X_r & w_r \end{bmatrix},$$

such that the following three conditions are satisfied.

i. Each state schema

$$\begin{bmatrix} v_i & Y_i \\ X_i & w_i \end{bmatrix}$$

in C is such that
(a) v_i is a node in EM",
(b) w_i is a node in EN", and
(c) X_i and Y_i are two (possibly infinite) sets of message sequences.
Each state schema can be viewed as a set of network states. A state

$$\begin{bmatrix} v_i & y_i \\ x_i & w_i \end{bmatrix}$$

is *in* some state schema

$$\begin{bmatrix} v_i & Y_i \\ X_i & w_i \end{bmatrix}$$

in C iff the message sequences x_i and y_i are in the sets X_i and Y_i, respectively.

ii. The initial state of [EM",EN"] is in some state schema in C.

iii. For every state s that is in some state schema in C, there exist two states s' and s" such that
(a) s" is in a state schema in C, and
(b) either (s' follows s over an edge in EM" and s" follows s' over an edge in EN") or (s' follows s over an edge in EN" and s" follows s' over an edge in EM").

For more details on how to apply these three techniques (network decomposition, machine equivalence, and closed covers) to the verification of the protocol models discussed in this paper, we refer the reader to [9] and [16].

6. Concluding Remarks

We have presented a formal model to define physical layer protocols and a methodology to verify the correctness of the resulting protocol models. We have applied this methodology to model and verify an asynchronous start-stop protocol and a synchronous transmission protocol with modems. In [9], we use the same formal model and its accompanying verification methodology to model and verify other physical layer protocols that include the EIA RS-449 Standard and the CCITT X.21 Recommendation.

We hope that this experience can inspire others to utilize these or other formal techniques in modeling communication protocols instead of resorting to informal descriptions that are ambiguous and usually invite errors or imprecision.

REFERENCES

[1] H. V. Bertine, "Physical interfaces and protocols," in *Computer Network Architectures and Protocols*, P. E. Green, Ed. New York: Plenum Press, 1982, pp. 57-83.

[2] G. V. Bochmann, "Finite state description of communication protocols," *Comput. Networks,* vol. 2, pp. 361-371, 1978.

[3] G. V. Bochmann and C. Sunshine, "Formal methods in communication protocol design," *IEEE Trans. Commun.*, vol. COM-28, pp.624-631, Apr. 1980.

[4] D. Brand and P. Zafiropulo, "On communicating finite-state machines," *Journal ACM*, vol. 30, pp. 323-342, Apr. 1983.

[5] C. H. Chow, M. G. Gouda, and S. S. Lam, "An exercise in constructing multi-phase communication protocols," in *Proc. SIGCOMM '84 Symposium*, June 1984.

[6] C. H. Chow, M. G. Gouda, and S. S. Lam, "A discipline for constructing multi-phase communication protocols," *ACM Trans. Comput. Syst.*, to appear Nov. 1985.

[7] R. J. Cypser, *Communications Architecture for Distributed Systems.* Reading, MA: Addison-Wesley, 1978.

[8] M. G. Gouda, "Closed covers: to verify progress for communicating finite state machines," *IEEE Trans. Software Eng.,* vol. SE-10, pp. 846-855, Nov. 1984.

[9] M. G. Gouda and K. S. The, "On modeling and verification of physical layer protocols," Tech. Rep., Dep. Comput. Sci., Univ. Texas, Austin, TX, in preparation.

[10] M. G. Gouda, K. S. The, and C. K. Chang, "Verification of distributed synchronization systems via decomposition," Tech. Rep., Dep. Comput. Sci., Univ. Texas, Austin, TX, in preparation.

[11] M. G. Gouda and C. H. Youn, "On the notion of equivalence for communicating finite state machines," *TR-84-14*, Dep. Comput. Sci., Univ. Texas, Austin, TX, May 1984. Revised Feb. 1985.

[12] B. T. Hailpern and S. S. Owicki, "Modular verification of computer communication protocols," *IEEE Trans. Commun.*, vol. COM-31, pp. 56-68, Jan. 1983.

[13] S. S. Lam and A. U. Shankar, "Protocol verification via projections," *IEEE Trans. Software Eng.,* vol. SE-10, pp.325-342, July 1984.

[14] J. E. McNamara, *Technical Aspects of Data Communication.* Maynard, MA: Digital Equip. Corp., 1977.

[15] A. S. Tanenbaum, *Computer Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[16] K. S. The, "A framework for formal modeling and verification of physical layer protocols," Master's thesis, Univ. Texas, Austin, TX, 1985.

[17] P. Zafiropulo, C. H. West, H. Rudin, D. Brand, and D. Cowan, "Towards analyzing and synthesizing protocols," *IEEE Trans. Commun.*, vol. COM-28, pp. 651-661, Apr. 1980.