Modeling and Verification of a Real Life Protocol Using Symbolic Model Checking

Vivek G. Naik and A. P. Sistla

Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, IL 60680

1 Introduction

As the computing systems have grown in size and complexity it has become necessary to develop automated methods for checking the correctness of such systems. *Temporal logic modelchecking* [2] is one of such automated methods for verifying properties of finite state systems. The practical applicability of the original modelchecking system was limited due to the state explosion problem. Recently many techniques have been developed to overcome the state explosion problem. One of the methods that has been finding much application is symbolic modelchecking [1, 8, 3, 6]. The symbolic modelchecking approach, implemented as the SMV system, uses BDDs for symbolically representing sets of states and the transition relation. This approach allowed the possibility of handling systems with extremely large state spaces.

In this paper, we show how symbolic modelchecking has been used to verify a real life protocol. Specifically, we have used SMV tool to model and verify IEEE 802.3 Etherenet CSMA/CD protocol with minimal abstraction. The Ethernet CSMA/CD protocol is a protocol that allows a set of computer systems connected over a local area network to communicate with each other. The major steps involved in using the SMV system for verification of the protocol were to correctly identify the processes within the protocol, to model them in the SMV toolkit, and to specify and verify the required properties of the protocol. Some design issues while modeling such a protocol are also dealt with in the research.

We have verified the protocol under the asynchronous and synchronous models. The major problems encountered in using the SMV system were in modeling of the following aspects associated with the protocol: the *channel*, *collision detection* and *carrier sensing*, *delay modeling* (delay is used in successive attempts after a collision using the exponential backoff approach) and *synchronization of transmitters and receivers.* We first modeled the protocol at much detail and checked the properties. Under these two models, we used progressive abstraction to reduce the number of variables in each transmitter and receiver, and thus reduce the time taken for modelchecking. We have verified many properties for different stations, for various values for different values of maximum number attempts and frame sizes.

This paper describes the appraoches employed in the verification purpose. The paper is organized as follows. Section 2 briefly describes the SMV system and

[1] This research is partially funded by NSF grant CCR-9212183.

the specification logic CTL. Section 3 gives an introduction to the Ethernet IEEE 802.3 protocol and a formal model of it. Section 4 specifies various problems encountered in modeling in SMV and how they were solved. It describes various components of the protocol. Section 5 lists various correctness properties of the protocol that were specified and verified using SMV, together with the times taken for verification. Section 6 contains concluding remarks.

2 The SMV tool

The inputs to the SMV system consist of the description of transition system modeling the concurrent system and a correctness specification. The correctness specification is a formula of the branching time temporal logic CTL (Computation Tree Logic) [2] . This logic allows the specification of various safety and liveness properties that are of interest to concurrent systems. CTL is a propositional branching time temporal logic.

The input language of SMV allows the description of the state transition system as modules. Each module has a set of parameters that can be instantiated and reused. Thus if the system has many similar components then all of them can be defined as instantiations of a single module definition. It also provides for a hierarchical description of the system. The data types available are Booleans, scalars and fixed arrays. The language allows a parallel assignment syntax. The reader is refered to the [7] for a detailed description of SMV.

3 Ethernet Protocol

3.1 Informal Description

Modern Computer Networks are designed in a highly structured way. A seven layered model was proposed by the International Standards Organization (ISO) as a first step towards international standardization. The layered approach has been taken with the fact that each layer provides some primary service to its upper layer thus making their implementations and design independent of the other layers as long as the services needed are provided. IEEE's 802 standard for local area networks is the key standard for LANs.

Description of the CSMA/CD LAN Protocol

The IEEE 802.3 (from now referred as ethernet) protocol is a Local Area Network(LAN) communication protocol. This standard covers the Physical Layer and Medium Access Control sublayer which is a part of the Data Link Layer. The Data Link Layer sits above the Physical Layer and provides services to the Network layer.

This protocol is based on the concept of ALOHA system developed in 1970s by Norman Abramson and his colleagues at University of Hawaii. The system gives an elegant method for the allocation of a shared channel by multiple users. The users share the single communicating channel. A user sends data in the form of a stream of bits which is called a frame. The basic idea of the original protocol is to let users transmit whenever they have data to send. If collisions occur then users try to transmit the data after a random delay. A much improved version of the above protocol is the CSMA/CD protocol (Carrier Sense Multiple Access wit Collision Detection) protocol. In this protocol, whenever a station wants to send data it first checks if the channel is busy (i.e. if any one else is currently transmitting); If the channel is not busy then it goes ahead with transmission of the data. If the channel is busy, then the station waits until the channel is idle and then transmits the data. Collisions can occur if two stations try to transmit at the same time or with in a short duration of time determined by the propagation delay of the channel. This protocol incorporates a collision detection mechanism. Whenever collisions occur, all transmitting stations are notified of the collision. Rather than finish transmitting their frames, which are irretrievably garbled anyway, the transmitting stations that detect collisions would abruptly stop transmitting and go onto a phase of post collision arbitration. This improves the overall performance.

Post collision operation

The first station to detect the collision aborts transmission and transmits a short noise burst. This noise signal is the jam signal which indicates to all the other stations that there has been collision. The station then waits for a random amount of time and repeats the cycle. After collision the time interval is divided into slots of a period of 2τ, where τ is the one way propagation delay of data transmission. The propagation delay is divided into a slot time of 512 bits (51.2 μsec).

After the first collision each station waits for 0 or 1 slot times before trying again. The number of slots for which a station is going to wait depends upon a random number selected by that station. If two stations pickup the same random number then there will be another collision in their next attempt at transmission. After the second collision the stations wait for a random period of between 0, 1, 2 or 3 slot times and try again. If a third collision occurs then the random number picked will a value between 0 to $2^3 - 1$. In general, after i collisions a random number between 0 to $2^i - 1$ is chosen. After 10 consecutive collisions have been reached then the randomization interval is frozen to a maximum of 1023 slots. After 16 collisions the controller gives up and reports the failure to the upper layer.

This algorithm of dynamically choosing the delay is called *Binary Exponential Backoff*. This mechanism ensures that the delay time adapts to the number of stations involved on collision. This mechanism ensures that collision is resolved in a reasonable interval if many stations collide.

The protocol as such doesn't provide any mechanism for acknowledgement of received frames. Thus the destination station should verify for the checksum and then send the acknowledgement frame if the data received is error free.

3.2 Formal Specification of the protocol

A formal specification of the Ethernet protocol is given in in [11]. In this model each station consists of a set of processes communicating through shared variables. Each process is modeled as a timed transition system with upper and lower bounds on each transition. This specification is much clearer and more readable than the informal specification of Ethernet in IEEE 802.3 [10, 4, 9]. We use this as the basis in our verification.

The various processes at each station and the data flow between these processes is given in figure 1. Each process performs a particular function. The arrows in the figure indicate the communication of variables that are being modified/shared by the processes. The LLC sublayer sends raw data frames to the MAC layer. In the model the actual frame is not sent but the same effect is achieved by the LLC layer just setting a variable to indicate that the MAC layer can start transmission. The MAC layer consists of the following subprocesses.

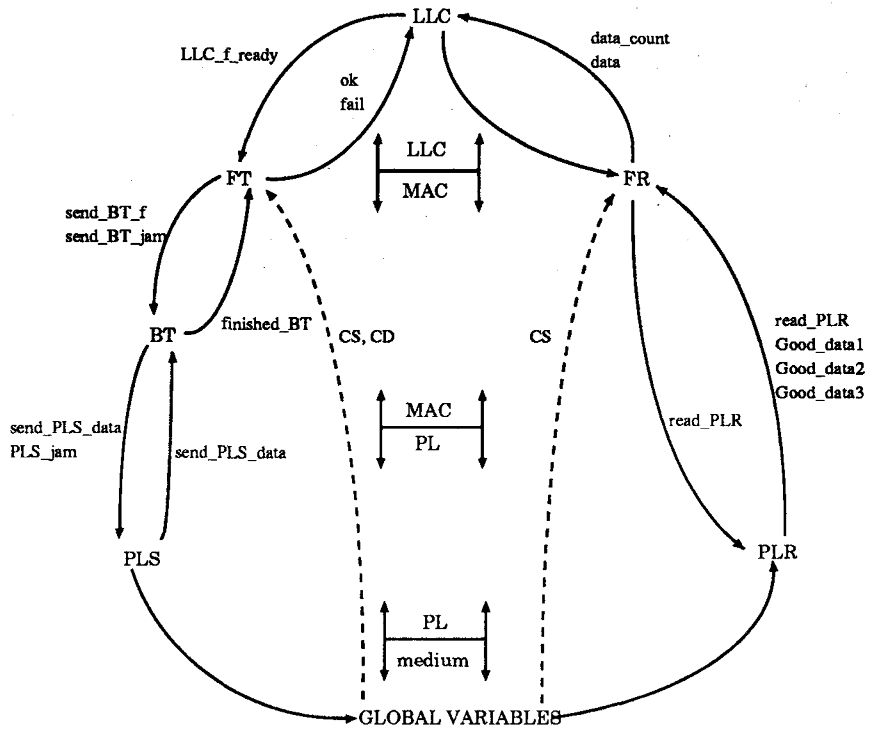

Fig. 1. Basic communication model of the Ethernet Protocol

1. *FT (Frame Transmitter)*: This process takes input from the variable *LLC_f_ready* to indicate that the frame is ready for transmission. It checks for availability of the carrier and if it finds it to be not busy, then it sends a *send_BT_f* signal to the bit transmitter. Then it waits for a *finished_BT* signal from the Bit Transmitter. While waiting, if it encounters a Collision Detection signal then

it sends a *send_BT_jam* signal to the BT and waits for *finished_BT* signal. If there is no collision detect signal till it receives the *finished_BT* signal then it sends an *ok* signal to the LLC indicating that the data transmission has been successful. If there had been a collision the process increments the variable indicating the number of attempts, picks up a random number according to the binary exponential backoff algorithm. It then sets the delay variable to the random number. After waiting for the amonut of time given by the delay, the process returns to the state in which it checks if the carrier is free. If collision occurs again then the process executes the above described steps till the number of attempts is less than maximum. Once the maximum limit is reached then the process terminates in the fail state indicating failure of transmission of the frame.

2. *BT (Bit Transmitter)* : This process is the link between the MAC layer and the processes in the Physical layer processes. The BT, upon receiving the signal *send_BT_f*, sends the *send_PLS_data* signal to the Physical Layer Sender (PLS) process to transmit the next bit from the data frame. If BT receives a *send_BT_jam* signal, then it indicates to the PLS to send the jam sequence bits. After completion of transmission of all the bits in the frame or the jam sequence, BT sends the *finished_BT* signal to the FT. This processes keeps track of the number of bits sent and the index of the next bit to be sent.
3. *FR (Frame Receiver)* : This process is the data receiver process of the MAC layer. It waits for the Carrier Sense signal to be true. It then communicates with the Physical Layer Receiver(PLR) and receives the data bits. The data frame received is then sent to the LLC layer.

The physical layer consists of the following processes:

1. *PLS (Physical Layer Sender)* : This process is the link between the transmission medium and the BT of the MAC Layer. Each station has a variable containing the next data bit to be transmitted. Upon receiving the signal from the BT to transmit data bit, PLS checks if the data to be transmitted is a jam bit or a data bit (this depends on the status of the *send_BT_jam* signal). Accordingly the data bit is set to one of {0,1,id} or to {J}. After the last data bit, an additional bit containing the value ND(no data) is sent; this provides spacing between successive frames transmitted on the channel. Other variables needed for synchronization of the Read and Write operations among all the stations are also set by this process. This topic is discussed in more detail in the Section 5.
2. *PLR (Physical Layer Receiver)* : This process is the receiver section of the Physical Layer. It receives data from the medium (we model them as global variables). When it receives a signal from the FR to read the data in, it reads it and stores it in the buffer. This process reads the data from the variable called the *chnl_data*. This variable is set using all the data bits of the transmitting stations in the model and its value represents the data on the channel. Extra variables are used to synchronize reading of the channel data with the sending of the data by the transmitting station.

The processes FT, BT and PLS form the transmitter part of a station. The processes PLR and FR form the receiver part of a station.

4 Modeling the protocol in SMV

The SMV uses for its input a transition based model of processes communicating to each other through shared and global variables. Thus, it is particularly suited for verifying the Ethernet protocol. However, the following major problems were encountered in modeling the protocol in SMV. Many of these problems are due to the lack of representation of time in SMV.

4.1 Issues in Modeling the protocol in SMV

1. *Channel representation*: The channel through which the stations communicate as we have seen in the previous sections consists of a stream of bits. These bits move at the speed of transmission that is 10 megabits per second. The propagation delay on the channel results in the stream of data bits to be unavailable at the same instant to all the stations on the network. To avoid modeling the propagation delay the data channel is assumed to be one bit long. This part of specification as given in [11] is a real time specification, and it needs to be modified so that it can be handled by SMV.
2. *Simulation of Transmission*: The rate at which the data is transmitted in the protocol is fixed, and the model in [11] achieves this by introducing a fixed time delay between the consecutive bits that are transmitted. The time delay between transmission of successive bits ensures that the receiving stations read the data before sending of the next bit. We achieve this by synchronizing the transmission and reception by using variable arrays.
3. *Processes with time constraints on their transitions*: Some of the processes have time bounds on their transitions. As in the case of transmission of data, the time bounds on the transitions of processes ensure that, for a shared variable between two processes, the previous value of the variable would have been read by the second process before the variable is updated again (most of the processes communicate data in a pattern in which the writing action of one process is followed by a reading action of the other). To get this same effect in the model without time bounds on the transitions it is necessary to validate that the reader has read the previous value before the writer writes again. This involves additional message passing. The technique used is similar to the transmission of data between stations and is discussed in the next section.
4. *The computation of Random Delay*: In the Binary exponential backoff delay mechanism after collision each station waits for a random amount of time slots. On the network it is imperative that if only one station picks up the smallest random number of all the stations involved in the collision, then it is bound to transmit successfully. To ensure that this behavior is preserved in the SMV model, it becomes necessary to have a centralized delay control

mechanism. This would take care of the post collision arbitration.
The following sections describe the basic modeling approach and the processes that effectively model the protocol.

4.2 Modeling the different Aspects in SMV

As indicated before, we considered two different method of modeling a set of stations using the Ethernet protocol. The first method is the asynchronous method where all stations and all processes in each station are modeled as asynchronous processes. The second method is where all stations are modeled as synchronous processes. In this method, the whole station is a single process.

Asynchronous Model

To enable communication between the processes belonging to a station module within themselves and with the other station modules in the model, we define global variables which are shared by all the processes. The variables include those which represent the status of data channel and those which are used for synchronization between the processes to ensure correct behavior.

Various components of the protocol are modeled as follows.

– *Data Channel modeling:* The data Channel was one of the complex aspects to model in the transition system. As mentioned in the section 4.1, some assumptions are made in modeling the channel. The most significant one is that the channel is assumed to be just one bit length, as opposed to that in real life as a stream of bits whose bit length depends upon the physical length of the cable. The data channel is assumed as a set of data bit variables with one data bit for each station on the network. Each Data variable is defined as follows:
$DATA : \{0, 1, 2, J, ND\}$;
These values represent the values that this variable takes. A value of { 0, 1, 2 } indicates good data bits. If the value of the data variable is J then it indicates that the station is writing jam sequence. The value of ND indicates that there is no data. The number of data variables depends on the number of stations in the system. Each station writes into the data variable which corresponds to its id number. The actual data on the channel is given by the variable *chnl_data* which is the composition of all the data variables. In a system with two stations the data on the channel is the composite of the data variables DATA1 and DATA2. The definition of the *chnl_data* is as given below.

```
chnl_data :=
   case
            DATA1 = ND  : DATA2;
            DATA2 = ND  : DATA1;
            1           : J;
   esac;
```

The above DEFINE takes care of the fact that the *chnl_data* has the valid data depending upon the value of the individual data variables. The third assignment sets it to J if the both of the data variables are not ND. Thus the *chnl_data* will always represent the composite of the two stations data bits.

- *Carrier Sense (CS) and Collision Detect (CD) variables*: The Carrier Sense (CS) variable which represents the status of the Channel is defined as follows:
  CS := !(DATA1 = ND & DATA2 = ND);
  This sets the variable CS to 1 if it is not the case that both the data variables are not ND. Thus a busy channel status is indicated by this variable when it is set.
  The Collision Detect (CD) variable which is set to 1 if there is collision on the channel. This is achieved by the following definition:
  CD := ! (DATA1 = ND | DATA2 = ND) | (DATA1 = J | DATA2 = J);
- *The Read Array to synchronize the Transmitters and the Receivers*: The data variables, mentioned above, are set by the transmitting side of the station, and the receiving side reads from the *chnl_data* variable. In the real life systems there is a time bound on each of the transitions which occur in the transmitting section and the receiving section. The rate at which the Receiver reads is same as the rate at which the Transmitter sends. Thus the Receiver will never read the same data bit twice from the channel and the Transmitter will not overwrite the last written data. Since we are modeling all processes as completely asynchronous and since we don not have time in our model, we achieve the synchronization between transmitters and receivers by using extra variables.

  We use an $n \times n$ array *Read* of binary variables to synchronize the transmitters and receivers. Transmitter i sets the bits in the i^{th} row to zero after writing on the channel and will not write again until all these bits are set to 1 by the receivers indicating that they have read it. Receiver set these bits to 1 after they read the data. Since all receivers run asynchronously we need one bit per receiver; Also, since the transmitters can try to transmit at the same time we need one row of bits per each transmitters. We also have used a model where only a one dimensional array of n bits are used. However, this model does not accurately depict the complete parallelism among the transmitters.
- *Modeling of Delay after collision:* The ethernet protocol uses a Binary Exponential Backoff algorithm for handling the post collision arbitration. When two stations get into collision then each one of them picks up a random number which depends upon the current attempt for sending data. Each station then waits for an amount of time given by the delay chosen by it, and tries to transmit again. Since the clocks at different stations are synchronized to run more or less at the same rate, the station to pick up the least delay will attempt to transmit first in the next try. Clearly, in an asynchronous model, we canot implement this by using an obvious count down of the delay vari-

able. We use a simple centralized delay monitoring mechanism for achieving the above effect.

Each station has its local variable *delay* which is set by the Frame Transmitter process. A value of 0 for this variable indicates that the corresponding station is not in an arbitration, i.e. not waiting to transmit. The Frame Transmitter process sets the *delay* variable according to the binary exponential backoff algorithm and waits for its *stn_go* signal to be set to true. The *stn_go* signal is globally defined for each station. This signal for station 1, in a two station configuration, is defined as follows.

stn1_go := !(stn1.FT.delay = 0) & !(stn2.FT.delay = 0) & (stn1.FT.delay <= stn2.FT.delay)

This definition ensures that the *stn1_go* is set to true only if both the *delay* variables are not zero and the third condition is true. The station which gets its *stn_go* signal to be true then goes ahead to transmitting data and sets the *delay* variable to 9 which then allows the other station to proceed. This further ensures that if this is the case only one station gets the *stn_go* signal then, it will transmit successfully next.

Synchronous Model

In the synchronous model, we do not need the *Read* array for synchronization purposes. Each value written by a transmitter in a clock cycle is read by all the receivers in the next clock cycle. No centralized delay monitoring system was needed. Essentially each transmitter decrements its delay counter in successive clock cycles, and when the delay becomes zero it tries to transmit again. Thus, this delay mechanism, in the synchronous model, is closer to reality than in the asynchronous model. However, in the synchronous model, each step of all the transmitters is synchronized which is not exactly the case in real life. These are the only differences between the asynchronous and the synchronous model.

5 Results and Inference

We were able to verify various properties in the asynchronous as well as the synchronous model. While testing the asynchronous model, we found some errors due to the fact that in the real life protocol there is an inherent assumption about the frame level synchronization which we did not model. We had to change our model appropriately to take this into consideration.

In the synchronous testing we used only one receiver since all the receivers are identical. This receiver is used to check that the transmitted frame is correctly received. We tested for different number of transmitters. We checked for the cases when the number of attempts is 2 and 4. We also checked for a frame size of 3 bits and of one bit. We checked for two properties. The first property asserts that whenever the LLC layer requests the transmission of a frame then eventually the MAC layer (i.e. frame transmitter) responds with a success or a failure message. This property for station 1, called property 1, is expressed by the following CTL formula:

$AG(LLCfready1 = 1 \rightarrow AF(ind1 = success \vee ind1 = fail))$.
Here *ind*1 is the variable through which the MAC layer sends a successful or failure message to the LLC layer. The amount of time taken for different parameter values is given in the table 1.

Table 1. Table of Results for synchronous models for checking property 1

Model Configuration			reachable	relation	time sec
transmitters	attempts	frame size	states	nodes	time
3	2	1	10K	13.3K	287
4	2	1	280K	34.9K	5839
3	4	1	77K	13.3K	3905
3	4	3	750K	140K	20,987

The second property that we checked asserts that the frame received by a receiver is the correct frame. In the synchronous model this is asserted as an invariance property. The following formula asserts this. In this, the predicate $trans1.FT.state1 = done$ indicates that transmitter 1 reaches a done state while the predicate *Gooddata*1 indicates that the frame buffer in the receiver denotes a good data frame from station 1.
$AG(trans1.FT.state = done \rightarrow Gooddata1)$
The timing results for property 2 are given in the table 2. The times given in this table include the time taken for computing the number of reachable states (usin the -r option) and this later time completely dominated the over all time taken for checking this assertion. When this option is removed the checking of this property was extremely fast. In fact, for the case of five transmitters the time reduced to 85 seconds when the -r option was not used.

In our asynchronous model, each station has all the processes on the transmitter side and the receiver side. Also each frame has three bits. We checked for the following properties. The first property (property 1) asserts that each station will eventually reaches a done state or a fail state. This property is asserted by the following CTL formula:
$AF(stn1.FT.state = done | stn1.FT.state = fail)$

The second property asserts that if a station reaches a success state all other stations must have received the frame sent by it. This is asserted by a CTL formula of the form $AG(Stn1.FT.state = done \rightarrow AF\ proper\text{-}reception)$
Here *proper-reception* is a state predicate on the receivers asserting that they received correct frame station 1. It is to be noted we needed an AF modality inside due to the asynchrony.

The third property we checked is that in a two station system whenever there

Table 2. Table of Results for synchronous models for checking property 2

Model Configuration			reachable	relation	time sec
transmitters	attempts	frame size	states	BDD nodes	time sec
3	2	1	10K	13.3K	23
4	2	1	280K	34.9K	247
5	2	1	7,500K	93.4K	3724
3	4	1	77K	13.3K	116
4	4	1	3950K	34.9K	2787
3	4	3	750K	140K	618

is a collision, then both stations reach a done state (successful transmission), or both states reach a fail state. This is expressed by a formula of the form $AF(p)$ where p is a state predicate.

The fourth property that we checked is that, in a two station system, in case of a collision the station picking up the lower delay will successfully transmit. This is expressed as a formula of the form
$AG(\mathit{station1\text{-}go} \rightarrow AF(\mathit{stn1.FT.state} = \mathit{done}))$

We checked the above properties for the asynchronous system with configuration of two and three stations in which one or two stations are active. A station is active if it is allowed to send messages. The results are given in table 3.

Table 3. Table of Results for the asynchronous model

Model Configuration		property	reachable	relation	time sec
stations	active stns	property	states	BDD nodes	time sec
2	2	1	701K	24K	51,500
2	2	2	701K	24K	52,000
2	2	3	701K	24K	59,466
2	2	4	701K	24K	62,400
3	2	4	1500K	64K	59,200

As can be seen from table 3 that verification of different properties in the asynchronous model, the modelchecking took more time. We believe that this is due to the inherent complexity of the protocol. We were able to reduce the times by 40%, for some of the cases of the table, by changing the variable ordering.

After modelchecking using a detailed model, we deleted lot of detail in the protocol and also removed the receiver part and modelchecked for the reduced system. In this system we checked for the property 1 given in the previous table. The number of reachable states reduced substantially and we were able to check for two station system with four attempts much faster, i.e. in 1600 seconds (approximately, 25 minutes).

6 Conclusion and Future work

In this paper, for the first time, we verified various properties of a detailed model of the Etherenet Protocol using symbolic modelchecking. The model for the protocol has been developed in stages, and the verification process identified some problems in our modeling. The difficulties in modeling were partly due to the absence of real time in the model checker. Solutions to these problems needed use of additional data structures, to preserve the correct behavior of the protocol. As an example, while developing the model it was found that the transmitter could possibly transmit before the receiver has reset its data buffers. This needed modifications in the model to transmit after all the receivers have reset.

The centralized delay monitoring mechanism used in the asynchronous model works for the case when there are two active stations. It can be modified to work for an arbitrary number of stations. In this case, system allows the station with smallest delay to transmit by setting appropriate flag, and decrements the delay variables of other waiting stations by th delay of the chosen process.

A more general approach for verifying properties of timed trantion systems under the discrete time model, is to transform the timed system into an untimed system by using extra time variables to keep track of the times for which each of the transitions has been enabled. In this case, we need to add another process/transition that models the clock and increments the time variables. The lower bounds bounds associated with each transition can be enforced by adding additional conjucts to the enabling conditions of the transitions. The upper bounds on the transitions are enforced by enabling the clock transition only when all the time variables obey the upper bounds of the corresponding transitions. Also, the time variables need to be reset to zero, in the action parts of each transition, whenever the corresponding transition is disabled or whenever the corresponding transition is executed. All of this can be done by a simple syntactic trnasformation of the transition system, and this can be automated. The SMV system can be used on the transformed system.

Our conclusion is that real life protocols can be verified using the SMV system. As indicated in the paper, our verification under the asynchronous model was done using a fairly detailed description of the system. The CTL specifications, which are used for model checking on the model, cover a wide range of properties. We believe that we can make the modelchecking faster by using dynamic variable reordering. This we expect to do in future.

Future work: The model which was developed is symmetric in the sense that there are more than one instances of the same module used in the model. Thus symbolic model checking with symmetry may make the verification faster. Secondly the SMV system presently doesn't have real time in its language. We feel that model checker for real time systems would model the problem more accurately.

References

1. J. R. Burch, E. M. Clarke, K. L. McMillan, D.L. Dill, "Symbolic Modelchecking: 10^{20} states and beyond", In Proceedings of fifth annual Symposium on Logic in Computer Science, June 1990.
2. E. M. Clarke, E. A. Emerson and A. P. Sistla. "Automatic verification of finite-state concurrent systems using temporal logic." *ACM Trans. Program. Lang. Syst. 8*, 2, (April 1986), 244-263.
3. E. M. Clarke, J. R. Burch, O. Grumberg, D. E. Long and K. L. McMillan. "Automatic verification of Sequential Circuit Designs." *Royal Society of London*, October 1991.
4. J. D. Day and H. Zimmerman. "The OSI reference model." In *Proc. of IEEE*, volume 71, pages 1334-1340, December 1983.
5. O. Lichtenstein and A. Pnueli. "Checking that finite state concurrent programs satisfy their linear specification." In *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, 1985.
6. K. McMillan and J. Schawalbe. "Formal verification of the Encore Gigamax cache consistency protocol." In *International Symposium on Shared Memory Multiprocessors.*, 1991.
7. K. L. McMillan. "The SMV system " February 1992.
8. K.L. McMillan. "Symbolic Modelchecking: An approach to state explosion problem" Ph.D. thesis, CMU-CS-92-131, may 1992.
9. A. Tanenbaum. *Computer Networks.* Prentice Hall, 2nd edition, 1989.
10. ANSI/IEEE std. *Information Processing Systems- Local Area Networks- Part3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.* The IEEE, Inc., NY, October 1991.
11. H. B. Weinberg and L. D. Zuck. "Timed Ethernet: Real-Time Formal Specification of Ethernet" In *Proc. 3rd CONCUR*, August 1992.