Specification and Verification of Collision-Free Broadcast Networks*

Pradeep Jain and Simon S. Lam

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

ABSTRACT

For high-speed local area networks that offer integrated services for data, voice, and image traffic, a class of demand-assigned multiple-access protocols have been presented in the literature. These protocols exploit the directionality of signal propagation and enforce time constraints to achieve collision-freedom. A correct implementation of such a protocol requires a careful analysis of time-dependent interactions of event occurrences using a formal method. To date, most protocol verification methods are intended for the analysis of asynchronous communication over point-to-point channels.

We present a model for broadcast bus networks. The novel features of our model include the ability to specify broadcast channels and the specification of real-time behavior. The broadcast characteristics of cables or buses are captured by some simple axioms. Real time is modeled as a discrete quantity using clocks and time variables. Real-time properties are specified by safety assertions. To illustrate our model and analysis method, we present a specification of the Expressnet protocol. We found that to achieve collision-freedom, a small modification to the original Expressnet protocol is needed.

1 Introduction

In a broadcast bus network, if two or more messages overlap in time at the same bus location, a collision results and the messages involved are garbled. Therefore, a multiple access protocol is needed to coordinate access to the bus. Recently, a class of demand-assigned multiple-access (DAMA) protocols has been proposed for broadcast bus networks [2,3,4,12,13]. These protocols incorporate some of the advantages of both ring networks and Ethernet. Stations are connected to a bus via taps. Since taps are passive elements, as opposed to the repeaters in a ring which are active elements, these networks are less susceptible to node and link failures. The protocols rely on observable channel events (such as the beginning or the end of a message, the bus becoming idle, etc.) for a station to determine when to transmit. Specifically, they exploit the directionality of signal propagation to provide collision-free access to a shared bus. Effectively, DAMA protocols employ a token-passing mechanism that is implicit and efficient.

Compared to the CSMA/CD protocol, DAMA protocols are capable of providing a higher channel throughput for high-speed LANs, as well as a bounded packet delay. Such schemes are thus particularly attractive for the next generation of LANs operating at speeds of 50-200 Mbps and providing integrated services (data, voice, video, facsimile, etc.). However, a correct implementation of any DAMA protocol will require a careful analysis of time-dependent interactions of event occurrences using a formal method.

DAMA protocols have two characteristics that distinguish them from protocols studied in the verification literature. First, the communication channel is a broadcast bus, i.e., a message transmitted by one station is received by all others. Also, multiple messages may coexist in the bus. These characteristics are fundamentally different from those of point-to-point links. Second, DAMA protocols are real-time protocols which require certain time-constraints to be met for their correct functioning [11]. These time constraints arise from the use of timeouts and imposition of bounds on delays, which are required to ensure collision-free access to the medium.

To model timing delays and constraints in a broadcast bus, the actual propagation of signals along the bus has to be modeled. Also, there could be multiple signals in different sections of the bus. The relative positions of these signals are also important. All these factors render the problem of modeling a broadcast bus unique. To date, almost all protocol verification techniques have focused on asynchronous communication over point-to-point channels [1,5,9,10,14].

While many DAMA protocols have been proposed in the literature, there is no formal model to specify them. Some authors have attempted to use finite state machines for specification. However, their representations are informal and too coarse [4,12]. That is, the states and state transitions are described with important details hidden. The analysis of these protocols, if any,

* This work was supported by National Science Foundation under grant no. NCR-8613338.

has been ad hoc. Typically, the reasoning is operational in nature and is based on time-space diagrams, which provide a graphical representation of the communication among the stations. Even the more 'formal' of these proofs essentially base their reasoning on a verbal description of the time-space diagrams. Since each such proof is carried out for a particular protocol and relies heavily on its operational details, it cannot be reused in the analysis of a different but similar protocol.

In [7], we presented a model for specifying broadcast bus networks and a method for verifying DAMA protocols. The novel features of this model include the ability to specify broadcast channels as well as the specification of real-time behavior. The model can be used to specify a variety of broadcast bus configurations. The timing relations for a particular broadcast bus configuration are proved independent of specific protocols. Furthermore, proofs of protocol properties are independent of the number of stations and their positions on the bus.

In this paper, the model is extended in two ways: (1) to account for the detection delay that is inherent in the sensing of the channel state by a station, and (2) to allow a station to perform concurrent actions. These extensions make possible the specification of a wider class of protocols. As an illustration, we specify the Expressnet protocol [12] formally. We show that for the protocol to be collision-free, a small modification to the original protocol is necessary.

In section 2, we describe our model for broadcast bus protocols. To illustrate the model, the Expressnet protocol (with a small modification) is specified in section 3. In section 4, we present our proof system. We use it in section 5 to show that the modified Expressnet protocol is collision-free. In section 6, we point out some special features of our model.

2 Modeling Broadcast Bus Networks

Several configurations and access methods are in use for DAMA networks [3]. We illustrate our model by using it to represent and verify a protocol based on the Unidirectional Bus System (UBS) configuration. (Our model is also applicable to the other broadcast bus network configurations [7].)

Unidirectional Bus System : The transmitted signal travels in only one direction. This directionality imposes an ordering on the stations. One way of achieving broadcast communication is to use two unidirectional buses, one in each direction. Another way is to fold the cable (figure 1). There are two logical channels, one inbound (for receiving) and the other outbound (for sending).

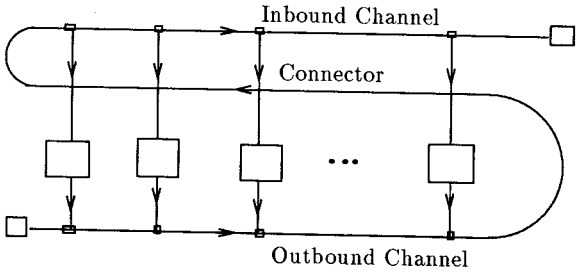

Figure 1. The UBS Configuration

2.1 The Model

The network is modeled by a system of concurrent processes. Each station attached to the bus is represented by a *station process*. In the UBS configuration with two buses, each bus is modeled by a *channel process*. In the UBS configuration using a folded cable, the bus is represented by two channel processes: one for the inbound segment and the other for the outbound segment. A host connected to a station is modeled by a *user process*.

Interprocess Communication

Interprocess communication is achieved by shared variables. A station or channel process has both local and shared variables. A user process has only shared variables (shared with a station process). Such a variable can be updated by the user process, thus causing a change in the state of the station process. Other than the specification of such state changes, user processes are not explicitly modeled.

A variable shared between a pair of processes (a station process and a channel process, or a station process and a user process, or a pair of channel processes) can be read by both processes. It can be written by one process or both processes. A shared variable that can be written by a single process only is said to be an *exclusive-write* variable of that process. A shared variable that can be written by both processes is said to be a *mutual-write* variable.

Modeling of Time

The access methods rely on the observance of certain time constraints imposed on the system. Real time is modeled by means of a global clock [1], time variables, and time events. Using the time variables, real time constraints, such as timeouts and bounded delays, can be specified. The clock and time variables are discrete, i.e., they can have only integer values. A time event corresponds to a clock tick and marks the passage of one time unit. The real time constraints and the time events impose a temporal ordering on the events of the collection of processes.

The value of each time variable is incremented with each tick of the global clock. A tick marks the end of a time slot, which is a time interval of one time unit in duration. Events take place only at ticks. An assertion made for time 'τ' is assumed to hold for the entire duration of the time slot immediately before the tick that sets the global clock to 'τ'.

In the following, the variable 'τ' refers to the value of the global time. We follow the convention that time variables, including 'τ', have the initial value 0.

Station Process

A station is specified by a set of local variables, a set of shared variables, and a program implementing the access algorithm. (The local variables may include time variables needed

[1] The assumption of a global clock can be relaxed. This is the topic of a forthcoming paper:

to implement timing constraints.) Shared variables are shared with a user process or a channel process. The language used is a subset of Pascal, augmented with three temporal primitives. The axioms for the procedural constructs (assignment, alternation and repetition) are the same as those described by Hoare in [6]. The temporal primitives are described below. These primitives specify the condition under which a process halts or resumes processing.

The predicate stated before a primitive is its pre-condition, and τ in the predicate refers to the global clock at that point in the program, i.e., before the execution of the statement. We will denote this value of the global clock as τ_{begin}. The predicate stated after a primitive is its post-condition, and τ in the predicate refers to the value of the global clock at that point, i.e., after the execution of the statement. This value of the global clock will be referred to as τ_{end}.

Let V be the set of exclusive-write variables of the station process, W be the set of mutual-write variables, and X be the set of shared variables (of the station process) that are exclusive-write variables of other processes. We shall also use the following notation:

$\{v_1, v_2, \ldots, v_m\}$	subset of V
$\{w_1, w_2, \ldots, w_n\}$	subset of W
$e_1, \ldots, e_{m+n}$	expressions
e (τ)	value of expression e at time τ
P	predicate over variables in V
C	predicate over variables in W $\cup$ X

1. **set**

In the following, $P^{v_1,v_2,\ldots,v_m}_{e_1(\tau),e_2(\tau),\ldots,e_m(\tau)}$ refers to the predicate P with all free occurrences of v_i replaced by $e_i(\tau)$.

$\{(\tau = \tau_{begin})$ **and** $P^{v_1,v_2,\ldots,v_m}_{e_1(\tau),e_2(\tau),\ldots,e_m(\tau)}\}$

set $v_1, \ldots, v_m, w_1, \ldots, w_n := e_1, \ldots, e_{m+n}$

$\{(\tau = \tau_{end} = \tau_{begin} + 1)$ **and** $P(\tau)\}$

The **set** command takes one clock tick to execute. It assigns e_1 to v_1, e_2 to $v_2, \ldots$, and e_n to v_n in one atomic operation.

2. **wait-seq**

Let $C_1, C_2, \ldots C_n$ be boolean conditions, and $T_1, T_2, \ldots, T_n$ be durations of time. For notational convenience, we define the following terms:

pattern $\equiv C_1$ **for** $T_1; C_2$ **for** $T_2; \ldots; C_n$ **for** T_n
$S = T_1 + \ldots + T_n$

The **wait-seq** statement causes the process to halt and remain idle until the 'pattern' described by the statement is observed in its entirety (wait-seq stands for wait-for-sequence).

The predicate match (pattern, t), defined below, is true at time t if there is a match for the pattern, i.e., condition C_1 is observed to be true for a time period T_1, immediately following which C_2 is seen to be true for T_2, and so on, and the pattern ends at time t.

match(pattern, t) $\equiv$
$\forall t_n : t - T_n < t_n \leq t : C_n(t_n)$
and
$\forall t_{n-1} : t - (T_n + T_{n-1}) < t_{n-1} \leq t - T_n : C_{n-1}(t_{n-1})$
$\vdots$
and
$\forall t_1 : t-(T_n+\ldots+T_1) < t_1 \leq t-(T_n+\ldots+T_2) : C_1(t_1)$

The axiom for the **wait-seq** states that during the execution of the wait statement, the exclusive-write variables of the process do not change. The **wait-seq** terminates at time τ_{end} if a match is found for the pattern at time τ_{end}, i.e., the pattern started some time after τ_{begin} and the first match for the pattern occurred at τ_{end}.

$\{(\tau = \tau_{begin})$ **and** $P(\tau)\}$

wait-seq (pattern)

$\{\tau = \tau_{end} \geq \tau_{begin} + S$ **and** $\forall t : \tau_{begin} \leq t \leq \tau_{end} : P(t)$
and match(pattern, τ_{end})
and $\forall t' : \tau_{begin}+S \leq t' < \tau_{end}$: **not** match(pattern, t')$\}$

3. **wait-par**

In the following, sequence$_i$ stands for a sequence of wait and Pascal statements. (The **set** statement is not allowed to appear in such a sequence.) Nesting of **wait-par** is permitted.

The following axiom defines the semantics of a sequence in terms of the semantics of its components, i.e., **wait-seq**, **wait-par** and Pascal statements.

$$\frac{\{\ P\ \}\ \text{sequence}_1\ \{\ P'\ \},\ \{\ P'\ \}\ \text{sequence}_2\ \{\ Q\ \}}{\{\ P\ \}\ \text{sequence}_1; \text{sequence}_2\ \{\ Q\ \}}$$

Given sequence$_1, \ldots,$ sequence$_n$ such that :

$\{\ P\ \}$ sequence$_i$ $\{Q_i\}$, for i = 1, 2, ..., n,

the axiom for the **wait-par** statement is the following:

$\{(\tau = \tau_{begin})$ **and** $P(\tau)\}$

wait-par
sequence$_1$; label := l_1
|| sequence$_2$; label := l_2
$\vdots$
|| sequence$_n$; label := l_n
end-wait-par

$\{(\tau = \tau_{end} > \tau_{begin})$ **and** $\forall t : \tau_{begin} \leq t \leq \tau_{end} : P(t)$
and label = $l_1 \Rightarrow Q_1$ **and** ... **and** label = $l_n \Rightarrow Q_n\}$

The meaning of the **wait-par** statement is the following. When control reaches the **wait-par** statement, start 'executing' all sequences concurrently (wait-par stands for wait-in-parallel). The **wait-par** terminates as soon as any one of the sequences completes execution (' fires'). When that happens, control passes to the statement following the **wait-par** statement, and all sequences inside the **wait-par** are terminated. Since the **set** statement is not allowed in any of the sequences, upon termination of the **wait-par** the process is in the same state as it would be in if it had executed only the firing sequence. Depending upon which of the sequences fired, the process will carry out a specific action. The variable 'label' is used as an auxiliary variable to identify the sequence that fired, and is used to select the piece of code to be executed after the **wait-par** statement. In this sense, the **wait-par** acts as a selection statement.

Consider the frequently used special case of **wait-par** where each sequence consists of a single **wait-seq**(pattern) statement. We define the following terms:

$\text{pattern}_i \equiv C_{i,1}$ **for** $T_{i,1}; \ldots; C_{i,n(i)}$ **for** $T_{i,n(i)}$
where n(i) is the number of terms in pattern_i,
and, $S_i = T_{i,1} + T_{i,2} + \ldots + T_{i,n(i)}$

We have the following axiom:

$\{(\tau = \tau_{\text{begin}})$ **and** $P(\tau)\}$

wait-par
wait-seq(pattern_1); label := l_1
|| **wait-seq**(pattern_2); label := l_2
⋮
|| **wait-seq**(pattern_m); label := l_m
end-wait-par

$\{(\tau = \tau_{\text{end}})$ **and** $(\forall t : \tau_{\text{begin}} \le t \le \tau_{\text{end}} : P(t))$
and
label = $l_1 \Rightarrow (\tau = \tau_{\text{end}} \ge \tau_{\text{begin}} + S_1$ **and** match $(\text{pattern}_1, \tau_{\text{end}}))$
⋮
and
label = $l_m \Rightarrow (\tau = \tau_{\text{end}} \ge \tau_{\text{begin}} + S_m$ **and** match $(\text{pattern}_m, \tau_{\text{end}}))$
and $\forall t_1 : \tau_{\text{begin}} + S_1 \le t_1 < \tau_{\text{end}}$: **not** match $(\text{pattern}_1, t_1)$
⋮
and $\forall t_m : \tau_{\text{begin}} + S_m \le t_m < \tau_{\text{end}}$: **not** match $(\text{pattern}_m, t_m)$ $\}$

The axiom states that the **wait-par** statement terminates as soon as any one of the patterns is matched by the input conditions. Note that two or more patterns may achieve a match at the same time, in which case a non-deterministic choice is made.

Special cases of wait statements

Some forms of the **wait-seq** and **wait-par** statements are used frequently. They are given special names.

1. **delay** (T)
   ≡ **wait-seq** (true **for** T)

   The statement **delay** (T) causes the process to halt and remain idle for time T. It takes time T to execute.

   $\{(\tau = \tau_{\text{begin}})$ **and** $P(\tau)\}$

   delay (T)

   $\{(\tau = \tau_{\text{end}} = \tau_{\text{begin}} + T)$ **and** $(\forall t : \tau_{\text{begin}} \le t \le \tau_{\text{end}} : P(t))\}$

2. **wait** (C)
   ≡ **wait-seq** (C **for** 1)

   The statement **wait** (C) causes the process to halt and remain idle until the specified boolean condition, C, is observed to be true. (If C is already true when the execution of **wait** (C) begins, the process still waits for the next time slot during which C is observed to be true.) The time taken for the **wait** (C) command is indeterminate, as it depends upon when C becomes true.

   $\{(\tau = \tau_{\text{begin}})$ **and** $P(\tau)\}$

   wait (C)

   $\{(\tau = \tau_{\text{end}} > \tau_{\text{begin}})$
   and $(\forall t : \tau_{\text{begin}} \le t \le \tau_{\text{end}} : P(t))$ **and** $C(\tau)\}$

In the **wait-seq** statement, if the time for a particular condition is 1, we will omit the '**for** T' clause, i.e., **wait-seq** (C_1 **for** 1; C_2 **for** T_2) will be written as **wait-seq** (C_1; C_2 **for** T_2).

Each program is essentially a loop which a station process executes continually. The statements within the loop involve computations on local and shared variables, and sensing and setting of signals on the channels (to be described later).

Channel Process

The channel process specification consists of a description of the state of the channel, and the transition rules describing how the channel state changes with time. A bus is divided into segments, the length of a segment being the distance the signal propagates on the bus in one time unit. The segments are numbered 0 to N, with the numbers increasing along the direction of the unidirectional bus. A station on the bus is referred to by its position on the bus, i.e., the station connected to the bus at position p will be called station p. (Therefore the station numbers may not be contiguous.)

The state of the channel consists of the state of each segment, i.e., the presence or absence of a signal in each segment. Thus the state of the channel at any given time is essentially described by specifying the portion(s) of the channel carrying a signal at that time. Transition rules are stated which describe how the

state of each segment changes with time. The state transition rules capture the propagation of signals, transmitted by stations, along the channel.

This simple model of a channel is actually quite general: it can model simultaneous transmission by two or more stations, and can model the presence of several messages in the channel. A variety of broadcast bus networks can be specified [7].

The UBS configuration for Expressnet (see section 3) is modeled by an outbound-channel process (for the outbound-bus) and an inbound-channel process (for the inbound-bus). The status of the outbound-channel is represented by an array called $\mathbf{c_{out}}$, and that of the inbound-channel by the array $\mathbf{c_{in}}$. There is one element in the array for each segment of a bus. $\mathbf{c_{out}}$ (p) represents the status of the outbound-bus at segment p, and $\mathbf{c_{in}}$ (p) that of the inbound-bus at segment p.

$\mathbf{c_{out}}$, $\mathbf{c_{in}}$: array [0..N] of 0..1
$\mathbf{c_{out}}$ (p) : 1, if a carrier is present on the outbound-bus at segment p
0, otherwise
$\mathbf{c_{in}}$ (p) : 1, if a carrier is present on the inbound-bus at segment p
0, otherwise

3 The Expressnet Protocol

In this section, we illustrate our model by specifying the Expressnet protocol [12]. First, we give a brief description of the Expressnet protocol in section 3.1. A formal specification follows in section 3.2.

3.1 Description of Expressnet

Topology of Expressnet

Expressnet is based on the UBS architecture (figure 1). Each station transmits on the outbound channel and receives on the inbound channel, and has the ability to sense the transmissions by stations on the upstream side of its transmitter. The inbound and outbound channels are connected by a *connector*. The end-to-end propagation delay along the inbound or outbound channel is denoted by τ. The propagation delay along the connector is τ_c (where $0 \leq \tau_c \leq \tau$). The propagation delay between the outbound and inbound taps for each station is fixed and equal to $\tau + \tau_c$. The detection-delay, i.e., the time required to detect the presence or absence of a carrier on a bus, is d.

Train of Transmission Units

A transmission unit (TU) consists of a **preamble** followed by the information **packet** itself. Information packets may be of fixed or variable size. The preamble is for synchronization at the receivers. It is sufficiently long for a receiver to detect the presence of the transmission unit, and to synchronize bit and packet boundaries. Stations transmit their TUs in a round-robin fashion. The succession of TUs transmitted in the same round is called a **train**.

Events $\mathrm{EOC_{out}}$ and $\mathrm{EOT_{in}}$

Functions c_{in} and c_{out} are defined to indicate the presence of a carrier on the inbound and outbound channels, respectively.

$c_{in}(p, t) = 1$, if station p detects a carrier on the inbound channel at time t
0, otherwise

$c_{out}(p, t) = 1$, if station p detects a carrier on the outbound channel at time t
0, otherwise

Function $\mathrm{train_{in}}$ is defined to indicate the presence of a train on the inbound channel.

$\mathrm{train_{in}}(p,t) = 1$, if station p detects a train on the inbound channel at time t
0, otherwise

$\mathrm{train_{in}}(p,t) = 1 \equiv c_{in}(p, t - d - 1) = 1$ or $c_{in}(p,t) = 1$

Event $\mathrm{EOC_{out}}(p,t)$, for end-of-carrier, is said to occur when $c_{out}(p,t)$ changes from 1 to 0. Event $\mathrm{EOT_{in}}(p,t)$, for end-of-train, occurs when $\mathrm{train_{in}}(p,t)$ changes from 1 to 0.

Elements of the Access Mechanism

The basic access mechanism followed by the Expressnet protocol is the **attempt-and-defer** method. As long as a train is in progress on the outbound channel, each station tries to transmit a TU using that method. Immediately following the detection of the $\mathrm{EOC_{out}}(p,t)$ event, station p starts transmission of its unit. Simultaneously, it monitors the outbound channel (on the upstream side of its transmission tap) for the presence of a carrier. If another station, p', with index lower than that of p, has also started transmission following its detection of $\mathrm{EOC_{out}}$(p', t), station p will detect a carrier within the first d seconds of its transmission. If that happens, station p immediately aborts its current transmission, and defers to the one from the upstream station. Otherwise, it completes the transmission of its unit.

Apart from the basic access mechanism, the protocol has two more components:

1. After all stations have had a chance to transmit a packet in a round, each station executes a procedure to start a new train. This consists of transmitting an unmodulated signal for duration d, called the **locomotive**, following the $\mathrm{EOT_{in}}(p,t)$ event. (By virtue of the Expressnet topology, locomotives transmitted by different stations overlap exactly.)

2. If a station determines that the network is asleep, it undertakes a **cold-start** procedure. This comprises the transmission of an unmodulated signal, called **pilot**, until a carrier is observed on the inbound channel.

3.2 Specification of Expressnet

Station process p shares variables **talk** (p) and $\mathbf{c_{out}}$ (p) with the outbound-bus process, and variable $\mathbf{c_{in}}$ (p) with the inbound-bus process. The inbound-bus and the outbound-bus processes share variable $\mathbf{c_{out}}$ (N).

When station p starts transmitting a signal on the bus, it sets variable **talk** (p) to true; **talk** (p) is set to false at the end of the transmission. **talk** (p) affects the state of the outbound-bus, and can be considered as output of the station process.

During its operation, station p continuously monitors the two buses, and takes actions based on the status of each bus. Therefore, $\mathbf{c_{in}}$ and $\mathbf{c_{out}}$ can be considered as inputs to the station process.

While executing the access protocol, the station process transmits other signals besides the information packet itself. For clarity and ease of stating and verifying the property of collision-freedom, it is useful to distinguish the various signals transmitted by the station: pilot, locomotive, preamble, and packet. We introduce four variables in the program to indicate the signal that is being transmitted. These variables are: **transmit-pilot**, **transmit-loco**, **transmit-preamble**, and **transmit-packet**. Each variable is set to true for the duration that the station is transmitting the corresponding signal. Note that the variable **talk** indicates the transmission of a signal of any type. Hence, **talk** is true if any one of the **transmit** variables is true.

The variable **packet-to-send** is shared between the station process and a separate user process. It is set to true by the user process whenever it needs to transmit a packet. It is set to false by the station process after the packet has been successfully transmitted. The variable **transmission-time** denotes the time taken for a packet transmission. Variables **request-to-wake-up** and **request-to-sleep** are also shared between the station process and the user process, and are set to true by the user process whenever it wishes the station process to wake up or sleep, respectively. These variables are set to false by the station process, once it wakes up.

Time Constraints

Since time is modeled as a discrete quantity, events can take place only at ticks. An action in response to a condition can start only at the beginning of the time slot following the one in which the condition becomes true. This introduces a delay of one time unit between a condition and its response. The protocol makes use of the following time constraints:

1. Two consecutive TUs are separated by a gap of duration d+1 time units, which is the time necessary to detect $\mathrm{EOC_{out}}$ and start transmission.
2. Event $\mathrm{EOT_{in}(p,t)}$ occurs when there are no more TUs in the current train, i.e., there is no signal on the inbound bus for d+2 time units.
3. The time gap between two consecutive trains, defined as the time between the end of the last TU in a train and the beginning of the locomotive of the subsequent train, is $\tau + \tau_c + 2d + 2$.
4. The net is asleep if there is no signal on the inbound bus for $\tau + \tau_c + 2d + 3$ time units.

3.3 Modification introduced to achieve collision-freedom

The Expressnet protocol as specified in [12] does not guarantee collision-free access to the bus. Consider the following scenario. Stations p_1, p_2 and p_3 are awake and have finished transmitting their packets in the ongoing round. In due course of time, each of them detects an $\mathrm{EOT_{in}}$ event and transmits a locomotive of length d units. Assume that p_1 and p_2 do not have a packet to transmit by the time they finish transmitting their locomotives. They will both stop transmission, and will start waiting for the next $\mathrm{EOT_{in}}$ or $\mathrm{EOC_{out}}$ event. If p_3 has a packet to send, immediately following the locomotive, it will proceed to transmit its preamble. At the end of the preamble, p_3 will detect the outbound bus to be idle, thereby concluding that no station upstream of it is going to transmit a packet in the current round. Therefore, it starts transmitting its packet.

Because of the detection delay, p_2 will detect the end of p_1's locomotive as an $\mathrm{EOC_{out}}$ event, d time units after it completes its own locomotive transmission. If by then it has received a packet to transmit, p_2 will start transmitting its preamble and packet, which will collide with p_3's packet. Clearly, this behavior is undesirable.

To achieve collision-freedom, we propose the following modification. Following the locomotive transmission, if a station does not have a packet to send, it waits for d+1 time units before checking for the next $\mathrm{EOC_{out}}$ or $\mathrm{EOT_{in}}$ event. This ensures that the end of the locomotive will not be seen as an $\mathrm{EOC_{out}}$ event by any station. Therefore, if a station receives a packet to transmit after it has had a chance to transmit in the current round, it will have to wait for the next train to start before it can transmit its packet.

It should be noted that the introduction of this wait does not increase the overall packet delay for any station. This is because the time spent in this wait is less than time until the next $\mathrm{EOT_{in}}$ event.

Station Program

The program for station p implementing the Expressnet protocol is given on the next page. The assertions appearing in the program are derived from the station process' axioms stated in section 2. Their meaning will be explained in section 5. (We have used the following abbreviations in the program annotations: *T* stands for *transmit*, and *C* stands for *clear*. For example, *T-packet* means *transmit-packet*, and *C-loco* stands for *clear-loco*.)

The initial value of all Boolean variables is **false**, and $\forall p : \mathbf{c_{in}}(p) = \mathbf{c_{out}}(p) = 0$.

```
wait (request-to-wake-up);
set request-to-wake-up, request-to-sleep := false, false;                    {t0 = t}

wait-par                    (* Check if net is awake *)
      wait-seq (cin = 0 for τ + τc + 2d + 3); label := net-asleep
    || wait (cin = 1 or cout = 1); label := net-awake
end-wait-par;

if label = net-asleep then                  {∀t' : 0 ≤ t' ≤ τ + τc + 2d + 2 : cin(p, t − t') = 0 ≡ C-pilot(p,t)}
    begin       (* Cold start *)                                  {t1 = t = t0 + τ + τc + 2d + 3}
        set transmit-pilot, talk := true, true;     {∃t1 < t : [C-pilot(p,t1) and cin(p,t − 1) = 0 and ∀t' : t1 < t' ≤ t : T-pilot(p,t')]}
        if cin ≠ 1 then wait (cin = 1);     {t ≥ t1 + 1 and cin(p,t − 1) = 0 and C-pilot(p,t1) and ∀t' : t1 < t' ≤ t : T-pilot(p,t')}
        set transmit-pilot, talk := false, false
    end;

repeat                                  {T-pilot(p,t) = T-loco(p,t) = T-preamble(p,t) = T-packet(p,t) = false}
    wait-par
        wait-seq (cin = 1; cin = 0 for d + 2); label := eotin                {EOTin(p,t)} ≡ {C-loco(p,t)}
      || wait-seq (cout = 1; cout = 0); label := eocout                                {EOCout(p,t) }
    end-wait-par;                                                                      {t2 = t }
                                                {label = eocout ⇒ EOCout(p,t) and label = eotin ⇒ EOTin(p,t) }

    if label = eotin then
        begin       (* Start a new train *)
            set transmit-loco, talk := true, true;                  { ∃t2 < t : [C-loco(p,t2) and ∀t' : t2 < t' ≤ t : T-loco(p,t')] }
            delay (d - 1)                  { ∃t2 < t : [C-loco(p,t2) and ∀t' : t2 < t' ≤ t : T-loco(p,t')] and t = t2 + d}
                                                                                                  ≡ {loco-sent(p,t)}
        end;

    if not packet-to-send then
        begin
            set transmit-loco, talk := false, false;
            delay (d + 1)       (* MODIFICATION : delay to ensure no station transmits out of turn *)
        end
    else                                                                                          { t3 = t }
        begin       (* Attempt to transmit packet *)          { EOCout(p,t) or loco-sent(p,t)} ≡ {C-preamble(p,t) }
            set transmit-loco, transmit-preamble, talk := false, true, true;
                                                                  { ∃t3 < t : [C-preamble(p,t3) and ∀t' : t3 < t' ≤ t : T-preamble(p,t')]}
            delay (d);                                 { ∃t3 < t : [C-preamble(p,t3) and ∀t' : t3 < t' ≤ t : T-preamble(p,t')]
                                                                                                  and (t = t3 + d + 1)}
            if cout = 1 then        (* Defer to upstream station *)
                set transmit-preamble, talk := false, false
            else
                begin       (* Complete transmission of TU *)                                {t4 = t = t3 + d + 1}
                                                  {∃t3 < t : [C-preamble(p,t3) and ∀t' : t3 < t' ≤ t : T-preamble(p,t')]
                                                                                  and (cout(p,t) = 0) ≡ C-packet(p,t)}
                    set transmit-preamble, transmit-packet := false, true;
                                                                  {T-pilot(p,t) = T-loco(p,t) = T-preamble(p,t) = false}
                                                          {∃t4 < t : [C-packet(p,t4) and ∀t' : t4 < t' ≤ t : T-packet(p,t')}
                    delay (packet-delay - 1);
                                                                  {∃t4 < t : [C-packet(p,t4) and ∀t' : t4 < t' ≤ t : T-packet(p,t')}
                    set transmit-packet, talk, packet-to-send := false, false, false
                end
        end
until request-to-sleep;
```

4 Verification

4.1 History Variables and Detection Delay

In section 2, the station axioms were stated in terms of conditions *as observed* by the station process. But there is an inherent delay between the time a condition *actually* becomes true, and the time it is *sensed* to become true by the station. For Expressnet, this time delay is the carrier-detection time, d, introduced in section 3.

The proof system will involve assertions on the past values of shared variables. Therefore it is convenient to define the following history variables, which also enable us to characterize the detection delay.

$c_{in,a}$ (p,t) = 1, if a signal is *actually* present at segment p of the inbound bus at time t
0, otherwise

$c_{out,a}$ (p,t) = 1, if a signal is *actually* present at segment p of the outbound bus at time t
0, otherwise

c_{in} (p,t) = 1, if a signal is *sensed* by the station at segment p on the inbound bus at time t
0, otherwise

c_{out} (p,t) = 1, if a signal is *sensed* by the station at segment p on the outbound bus at time t
0, otherwise

talk (p,t) = true, if there is a station at segment p and it is transmitting a signal at time t
false, otherwise

The following axioms relate the actual and sensed signals :

A1. $c_{in}(p,t) = c_{in,a}(p, t - d)$

A2. $c_{out}(p,t) = c_{out,a}(p, t - d)$

4.2 Axioms for the Channel Processes

The following axioms define the state transition rules for the channel processes.

1. For the outbound bus :

A3. $\forall p : 0 < p \leq N, \forall t > 0 : (c_{out,a}(p,t) = 1) \equiv (c_{out,a}(p-1, t-1) = 1)$ or $\text{talk}(p-1, t-1)$

A4. $\forall t : c_{out,a}(0,t) = 0$

2. For the inbound bus :

A5. $\forall p : 0 < p \leq N, \forall t > 0 : c_{in,a}(p,t) = c_{in,a}(p-1, t-1)$

A6. $\forall t : c_{in,a}(0,t) = c_{out,a}(N, t - \tau_c)$

4.3 Basic Theorems for Channel Processes

Using the axioms stated above, the following theorems can be proved. These theorems state timing relations between events at different points along the channel and their effects at remote points. These theorems are used, together with the invariant assertions to be proved for the station process, to verify system properties.

Theorem 1 : The following relations hold for the outbound-channel process :

a. $\text{talk}(p_1,t) \Rightarrow \forall p_2 > p_1 : c_{out}(p_2, t + p_2 - p_1 + d) = 1$

b. $c_{out}(p_1,t) = 1 \Rightarrow \exists p_2 < p_1 : \text{talk}(p_2, t - (p_1 - p_2) - d)$

The theorem states that for every time instant that station p_1 transmits a signal on the outbound-bus, each station to its right, p_2, will detect a signal after a delay which is equal to the propagation delay between the two stations plus the detection delay. Similarly, if station p_1 detects a signal, then some station to its left, p_2, must have transmitted a signal at a time which is (propagation delay + detection delay) before this particular time instant.

Theorem 2 : The following relations hold for the inbound-channel process :

a. $c_{in}(p_1,t) = 1 \Rightarrow \forall p_2 : c_{in}(p_2, t + p_2 - p_1) = 1$

b. $c_{in}(p_1,t) = 0 \Rightarrow \forall p_2 : c_{in}(p_2, t + p_2 - p_1) = 0$

The theorem states that all stations see the same signal on the inbound bus.

Theorem 3 : The following relations hold between the signals on the inbound and outbound buses:

a. $c_{out}(p_1,t) = 1 \Rightarrow \forall p_2 : c_{in}(p_2, t + (\tau + \tau_c) + (p_2 - p_1)) = 1$

b. $\text{talk}(p_1,t) \Rightarrow \forall p_2 : c_{in}(p_2, t + (\tau + \tau_c + d) + (p_2 - p_1)) = 1$

The theorem relates the time when a signal is present on the outbound bus or when a station transmits a signal to the time that signal is detected on the inbound bus.

Proofs of these theorems are given in [8].

The stations interact with each other only indirectly via the inbound and outbound buses. The effect of the activity of one station (starting or stopping transmission) is seen by each of the other stations as a change in the status of the buses. This effect is felt at the other station after a time delay which is equal to the propagation delay between the two stations. Our model captures these timing relations.

4.4 Proving Properties

A property of the system is stated as a predicate over the variables of the protocol system. To verify a given property, we prove invariant assertions for the station program using axioms for the station process presented in section 2. Timing relations are captured by Theorems 1,2 and 3. The proof consists of showing that the invariant assertions, together with the three theorems, imply the system property.

5 Proof of Collision-freedom

To prove that Expressnet is collision-free, we make use of several predicates, which are defined below.

Definitions

D1. $EOC_{out}(p,t) \equiv c_{out}(p,t-1) = 1$ and $c_{out}(p,t) = 0$

The predicate EOC_{out} corresponds to the end-of-carrier event, and is true when c_{out} undergoes a transition from 1 to 0.

D2. $EOT_{in}(p,t) \equiv c_{in}(p,t-d-2) = 1$
and $\forall t' : 0 \leq t' < d+2 : c_{in}(p,t-t') = 0$

The predicate EOT_{in} corresponds to the end-of-train event. It becomes true when c_{in} changes from 1 to 0 and remains 0 continuously for d + 2 seconds. This is because the gap between two consecutive TUs is d+1, and the station has to wait for an additional time unit to determine that no more packets are going to arrive in the current train.

D3. clear-pilot (p,t) $\equiv \forall t' : 0 \leq t' \leq \tau + \tau_c + 2d + 2 :$
$c_{in}(p,t - t') = 0$

The predicate **clear-pilot** stands for "clear to transmit pilot", and represents the condition which must hold for a station to transmit a pilot. It is clear to transmit a pilot when the station has observed the inbound channel idle for time $\tau + \tau_c + 2d + 3$ continuously. This is because the gap between consecutive trains is $\tau + \tau_c + 2d + 2$, and waiting for the above time period without any signal on the inbound bus ensures that the net is asleep.

D4. clear-loco (p, t) $\equiv EOT_{in}(p,t)$

The predicate **clear-loco** is true when it is clear to transmit a locomotive. That condition is met when an end-of-train is seen on the inbound channel.

D5. loco-sent (p, t) $\equiv$ clear-loco (p, t − d)
and $\forall t' : t - d < t' \leq t :$ transmit-loco (p, t')

clear-preamble (p, t) $\equiv EOC_{out}(p,t)$ or loco-sent (p, t)

The predicate **clear-preamble** describes the condition when it is clear to transmit a preamble. It is true when an end-of-carrier is seen on the outbound bus, or a locomotive has been transmitted following an EOT_{in}.

D6. clear-packet (p, t) $\equiv (EOC_{out}(p,t-d-1)$
or $EOT_{in}(p,t-2d-2))$ and $c_{out}(p,t) = 0$

The predicate **clear-packet** describes the condition which must hold for a station to transmit a packet. It is true when a preamble of length d + 1 has been transmitted, and $c_{out} = 0$, indicating that no station from upstream is transmitting a packet.

Invariants

The following assertions hold invariantly for the station program of Expressnet.

I1. talk (p,t) $\equiv$ transmit-pilot (p, t) or transmit-loco (p, t) or
transmit-preamble (p, t) or transmit-packet (p, t)

The above assertion states that the predicate **talk** is true as long as the station is transmitting any signal.

I2. transmit-pilot (p, t) $\Rightarrow$
$\exists t_0 < t :$ [clear-pilot(p,t_0) and $c_{in}(p,t-1) = 0$
and $\forall t' : t_0 < t' \leq t :$ transmit-pilot (p, t')]

The above assertion states that if station p is transmitting a pilot at time t, then the condition to transmit a pilot must have been true at an earlier time, and station p has been transmitting continuously since then.

The following three invariants are similar to **I2**, and apply to the transmission of locomotive, preamble and packet, respectively.

I3. transmit-loco (p, t) $\Rightarrow \exists t_0 < t :$ [clear-loco(p,t_0)
and $\forall t' : t_0 < t' \leq t:$ transmit-loco (p, t')]

I4. transmit-preamble (p, t) $\Rightarrow \exists t_0 < t :$ [clear-preamble(p,t_0)
and $\forall t' : t_0 < t' \leq t:$ transmit-preamble (p, t')]

I5. transmit-packet (p, t) $\Rightarrow \exists t_0 < t :$ [clear-packet(p,t_0)
and $\forall t' : t_0 < t' \leq t:$ transmit-packet (p, t')]

The annotated program for the Expressnet protocol is given in section 3. It can easily be checked that the above assertions are indeed invariant.

Condition for Collision-freedom

The pilots, locomotives and preambles from two or more stations may overlap in time. But the packet transmitted by a station should not be corrupted by any other signal. This condition is stated as the following assertion:

CF. $\forall t, \forall p_2 > p_1 :$
[transmit-packet$(p_1,t) \Rightarrow$ not talk$(p_2, t + p_2 - p_1)$
and talk$(p_1,t) \Rightarrow$ not transmit-packet$(p_2, t + p_2 - p_1)$]

A proof that the Expressnet protocol satisfies the above assertion is given in [8].

6 Some Features of Our Model

In our method, proving a system property comprises three steps : deriving the timing relations (Theorems 1, 2 and 3 of section 4.3), proving invariants for the station program, and showing that the system property follows from the timing relations and program invariants.

1. The theorems for the channel processes stated in section 4 essentially state the timing relations between events at a station and their effects at remote points in a simple and concise manner. It is possible to state the timing relations in this manner because of the way the channel is being modeled. Treating time as a discrete quantity enables us to treat the channel as made up of an integral number of

segments, which permits inductive reasoning. We are thus able to identify the portions of the channel that are busy at any time, and track the propagation of the signal along the channel.

The channel processes, and hence the timing relations, remain unaltered for different protocols based on the same configuration. Thus this method has more general applicability than other attempts at analyzing DAMA protocols.

2. The invariants of interest for the station program are the assertions that hold during transmission (in our example, assertions I1–I5 in section 5). These invariants capture the essence of the multiple-access algorithm followed by each station. These assertions involve only one station – the interaction among the stations has already been captured by the timing relations stated earlier.

   These invariants can be proved by examining just one process – that of a station. Thus the problem reduces to a verifying a single program in isolation. Checking for non-interference is not required, even though shared variables are being used. This is because the channel processes do not interfere with a station process, and the assertions used in the verification of a station process refer only to its own state and the *past* state of channel processes. The station process does interfere with the channel processes; the effect of that interference is incorporated in the channel axioms.

3. The proof is independent of the number of the stations and their positions in the network. The specification of the station process and the channel axioms capture all the relevant physical details.

7 Conclusions

DAMA protocols are particularly suitable for the next generation of LANs, which will operate at very high speeds and will offer integrated services for data, voice, video and facsimile traffic. These protocols exploit the directionality of signal propagation and implement stringent timing constraints to achieve collision-freedom. Correct implementation of these protocols will require a careful analysis of time-dependent interactions between event occurrences using a formal method. To date, most protocol verification methods have been focused on asynchronous communication over point-to-point links.

We have proposed a model for verifying real-time protocols for high-speed LANs. The novel features of our methodology are the ability to model broadcast channels as well as the specification of real-time protocol behavior. The method has potential applicability to a wide class of broadcast bus networks. A formal proof technique is used for analysis, instead of resorting to informal arguments. We illustrate our model and analysis method by specifying the Expressnet protocol and proving it to be collision-free, given a small modification to the original protocol.

References

[1] G. V. Bochmann, "Finite state description of communication protocols," *Computer Networks*, vol. 2, pp. 361-372, October 1978.

[2] K. Eswaran, V. C. Hamacher and G. S. Shedler, "Collision-free access control for communication bus networks," *IEEE Transactions on Software Engineering*, vol. SE-7, pp. 574-582, November 1981.

[3] M. Fine and F. Tobagi, "Demand assignment multiple access schemes in broadcast bus local area networks," *IEEE Transactions on Computers*, vol. C-33, pp. 1130-1159, December 1984.

[4] L. Fratta, "An improved access protocol for data communication bus networks with control wire," *Proceedings of the ACM SIGCOMM Symposium*, Austin, TX, March 1983.

[5] B. Hailpern and S. Owicki, "Modular verification of computer communication protocols," *IEEE Transactions on Communications*, vol. COM-31, Jan. 1983.

[6] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of ACM*, vol. 12, pp 571-580, 1969.

[7] P. Jain and S. S. Lam, "Modeling and verification of real-time protocols for broadcast networks," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 924-937, August 1987.

[8] P. Jain and S. S. Lam, " Specification and verification of collision-free broadcast networks," Tech. Report, Dept. of Computer Sciences, University of Texas at Austin (in preparation).

[9] S. S. Lam and A. U. Shankar, "Protocol verification via projections," *IEEE Transactions on Software Engineering*, vol. SE-10, pp. 325-342, July 1984.

[10] A. U. Shankar and S. S. Lam, "An HDLC protocol specification and its verification using image protocols," *ACM Transactions on Computer Systems*, vol. 1, pp. 321-368, Nov. 1983.

[11] A. U. Shankar and S. S. Lam, "Time-dependent distributed systems: Proving safety, liveness and real-time properties," *Distributed Computing*, vol. 2, pp. 61-79, 1987.

[12] F. Tobagi, F. Borgonovo and L. Fratta, "Expressnet: A High-performance integrated-services local area network," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, pp. 898-912, Nov. 1983.

[13] C. Tseng and B. Chen, "D-net: A new scheme for high data-rate optical local area networks," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, pp. 493-499, November 1983.

[14] P. Zafiropoulo et al, "Towards analysing and synthesising protocols," *IEEE Transactions on Communications*, vol. COM-28, pp. 655-660, April 1980.