SPECIFICATION OF REAL-TIME BROADCAST NETWORKS

Pradeep Jain and Simon S. Lam

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

TR-91-11 April 1991

Specification of Real-Time Broadcast Networks *

Pradeep Jain and Simon S. Lam

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

January 14, 1991

Abstract

We present a model for specifying real-time protocols that execute on broadcast bus networks. Protocol entities interact by sending and receiving binary signals on buses. The actual propagation of these signals is captured in our model by a set of channel axioms. Protocol entities are specified by sequential programs. The semantics of a set of programming constructs, including two novel *wait* constructs, are defined. To illustrate our model and verification method, we present a specification of the Expressnet protocol which was designed for collision-free access to a unidirectional bus. We discovered a scenario in which collisions can occur in the original Expressnet. To guarantee collision-freedom, we provide a modification to the protocol. The modified protocol is shown to be collision-free. We also derive a bound for its access delay.

Index terms: Specification, verification, real-time systems, broadcast networks, programming constructs, formal semantics, timed reachability graph, interval formulas.

*This work was supported in part by a grant from the Texas Advanced Research Program and in part by National Science Foundation under grant no. NCR-8613338 and under grant no. NCR-9004464.

1 Introduction

To date, almost all protocol specification and verification methods have been designed for protocols that employ message-passing over point-to-point communication channels [1, 6, 7, 12, 13, 14, 15, 17, 21]. We present in this paper a model for specifying protocols that execute on broadcast bus networks. Specifically, we are interested in protocols that rely upon the implementation of timing constraints to provide their services—we refer to these as *real-time protocols.*

Our work is motivated by high-speed broadcast bus networks for the transport of data, voice and image traffic. While many efficient multiple-access protocols have been proposed for these networks [3, 4, 5, 19, 20], we are not aware of any model designed primarily for their formal specification and verification. Some authors have used state transition diagrams for specifying broadcast bus networks; such specifications are informal and too coarse [5, 19], that is, many states and state transitions are described with important details hidden. Analyses of these protocols have been ad hoc. Typically, operational reasoning is used in the analyses. Proofs are based upon space-time diagrams that provide graphical representations of signal propagations on a bus. Because each proof is carried out for a specific protocol and relies heavily on the protocol's operational details, it is generally not reusable for a variation of the same protocol.

In our model, the communication mechanism and protocol entities, called *stations*, are specified separately. The actual propagation of binary signals on a bus is modeled explicitly by a set of channel axioms; the basic assumption is that signals propagate at a constant speed. The semantics of a set of programming constructs are defined. Two novel *wait* constructs are introduced, namely: **wait-seq**, waiting for a sequence of conditions, and **wait-par**, waiting for sequences of conditions in parallel.

We present a method for proving that protocols specified in our model have desirable properties. Our method is based upon the construction of a timed reachability graph (TRG). Each node in the graph is a *global formula* that is the conjunction of a set of *interval formulas.* An interval formula represents a description of a station's state variables over a time interval. We allow the time intervals of interval formulas in the same global formula to be different. To reduce the size of an actual TRG representation, we introduce some interesting aggregation techniques. We also use induction in our TRG construction so that the number of stations can be parameterized.

To specify that a protocol makes progress, we make use of the fact that there is a global clock variable in our model. Instead of temporal logic assertions, we use real-time assertions of the form: $P(i, t_1) \Rightarrow \langle \exists t_2 : 0 \le t_2 - t_1 \le \Delta : Q(j, t_2) \rangle$, where P and Q denote interval formulas, i and j station indices, t_1 and t_2 time parameters and Δ a time constant. A protocol property is verified by showing that it is satisfied for all $t_1 \ge t_0$, where t_0 is the starting time of protocol execution.

To illustrate our specification model and proof method, we specify the Expressnet protocol [19] which was designed to provide stations with collision-free access to a unidirectional bus. In applying our proof method, we found a scenario in which the original Expressnet is susceptible to collisions. We introduce a protocol modification and show that the modified protocol is collision-free. We also derive a bound on the access delay of each station.

Our model can be used for specifying a variety of broadcast bus configurations. In this paper, the unidirectional bus configuration of Expressnet is used throughout for illustration. It is not hard to see that other network configurations with multiple buses can be specified by modifying the channel axioms (e.g., see [10]). In fact, *the assumption of broadcast is not necessary.* For point-to-point channels, channel axioms can be specified as special cases of the axioms for a unidirectional bus (given in Section 4.2). There is also no restriction on the network topology.

The rest of this paper is organized as follows. In Section 2, we describe our specification model. In Section 3, we introduce and specify the original Expressnet protocol. In Section 4, we present a specification of the Expressnet unidirectional bus. In Section 5, we present our proof method. In Section 6, we describe a scenario of possible collisions in the original Expressnet; we then present a protocol modification, together with safety and progress properties of the modified protocol. Our conclusions are in Section 7. Formulas that are the "aggregate" nodes of a TRG constructed for the modified Expressnet protocol are given in the Appendix.

2 Model and Language Constructs

A network is modeled by a set of concurrent processes. There are three kinds of processes: (1) *station processes* representing protocol entities, (2) *channel processes* representing the communication mechanism, and (3) *user processes* representing users of protocol entities.

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, a station process and a user process, or a pair of channel processes) can be read by both processes, but can be written by one process only. A shared variable that a process can update is said to be a *write* variable of that process. A shared variable that a process can read but not write is said to be a *read-only* variable of that process.

Modeling of Time

The protocols of interest to us rely on the implementation of certain time constraints by the system. Real time is modeled by means of a global clock variable, time variables, and a time event. Using the global clock and 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 only have integer values. Occurrence of the time event corresponds to a clock tick that increments the global clock variable by 1.

Station Process

A station process is specified by a set of local variables, a set of shared variables, and a sequential program. The shared variables can be considered to be "communication variables"; interactions among station processes depend upon how the values of these variables change over time. That is, when one of these variables is updated by a station process, it may affect the state of another station process at a later time; this is the only way station processes communicate. These variables are divided into two sets: WV (the set of write variables) and RO (the set of read-only variables). The set of local variables, denoted by LV, is used for computation internal to the station. Some variables in the set LV $\cup$ WV may be auxiliary variables used solely for stating and verifying properties. (For a definition of auxiliary variables, see [13].)

Notation

LV	set of local variables
WV	set of write variables
RO	set of read-only variables
$\{x_1, x_2, \ldots, x_k\}$	subset of LV
$\{y_1, y_2, \ldots, y_m\}$	subset of WV
$e_1, e_2, \ldots, e_m$	expressions
τ	global clock variable
$t, t', t_1, t_2, \ldots, u, u_1, u_2, \ldots$	time parameters
$T, T_1, T_2, \ldots, \Delta, \Delta_1, \Delta_2, \ldots$	integer-valued constants or parameters

Timed and Interval Formulas

A *state predicate* of station j is a predicate over the state variables of station j, namely: its local and shared variables, its program counter, and the global clock variable τ. Let P (j) denote a state predicate of station j. From P (j), we define the following *timed formula*:

$$P(j, t) \equiv (\tau = t) \Rightarrow P(j)$$

The formula P (j, t) is true if and only if P (j) is true whenever the global clock value is t. (For notational convenience, we use the same name for the timed formula as that of the state predicate from which the formula is defined.)

An *interval formula* is defined from timed formula P (j, t) as follows:[1]

$$F(j, t) \equiv \langle \forall t' : t + \Delta_1 \leq t' \leq t + \Delta_2 : P(j, t') \rangle$$

For given values of t, Δ_1 and Δ_2, the formula F (j, t) is true if and only if P (j) is true whenever the global clock value is in the time interval $[t + \Delta_1, t + \Delta_2]$. Note that every timed formula is an interval formula (where $\Delta_1 = \Delta_2 = 0$).

Other formulas are constructed from interval and timed formulas using the logical operators: $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation).

Control Predicates

Instead of using a program counter explicitly, we make use of three control predicates to indicate the *control location* within a program. These control predicates are defined in the notation of [16]. In each station program, certain statements are labeled. Such labels are surrounded by curly brackets, "{" and "}." In the following, S refers both to a label and its associated statement and j denotes a station process. The control predicates, **at**, **in** and **after** have the following meanings:

at (j, S) : true iff control for station j's program is just before statement S
(S is the next statement to be executed)

in (j, S) : true iff control for station j's program is inside S (S is being executed)

after (j, S) : true iff control for station j's program is immediately after statement S

[1] An interval formula is different from the state predicate of Jahanian and Mok because the notion of "system state" is not used in [9].

(S has just finished executing)

Control Axioms

Timed formulas **at** (j, S, t), **in** (j, S, t) and **after** (j, S, t) are defined from the control predicates **at** (j, S), **in** (j, S) and **after** (j, S), respectively. In our model, it is possible for a statement to take zero time to execute; thus it is possible for **at** (j, S, t) and **after** (j, S, t) to be both true.[2] Timing relations among the three control predicates are stated by the following axioms. Axioms A1–A4 describe the progression of control during a statement's execution. Axiom A5 states that if a statement is currently executing, then control must have been just before that statement at an earlier time or the current time.

A1. $\mathbf{at}(j, S, t) \Rightarrow \mathbf{in}(j, S, t) \vee \mathbf{after}(j, S, t)$
A2. $\mathbf{in}(j, S, t) \Rightarrow \mathbf{at}(j, S, t) \vee \mathbf{in}(j, S, t-1)$
A3. $\mathbf{in}(j, S, t) \Rightarrow \mathbf{in}(j, S, t+1) \vee \mathbf{after}(j, S, t+1)$
A4. $\mathbf{after}(j, S, t) \Rightarrow \mathbf{at}(j, S, t) \vee \mathbf{in}(j, S, t-1)$
A5. $\mathbf{in}(j, S, t) \Rightarrow \langle \exists t' : t' \leq t : \mathbf{at}(j, S, t') \rangle$

We next define axiomatically language constructs for specifying station processes. Below, j denotes a station process, and S, S_1, S_2, ...denote statement labels in j's program. C (j) denotes a state predicate of station j, and C (j, t) is the timed formula defined from it.

Pascal-like Constructs

1. {S} $S_1; S_2$ (sequence of statements)

   A6. $\mathbf{after}(j, S_1, t) \Leftrightarrow \mathbf{at}(j, S_2, t)$

2. {S} **begin** $S_1; S_2; \ldots; S_n$ **end** (compound statement)

   A7a. $\mathbf{at}(j, S, t) \Leftrightarrow \mathbf{at}(j, S_1, t)$
   A7b. $\mathbf{after}(j, S_n, t) \Leftrightarrow \mathbf{after}(j, S, t)$
   A7c. $\mathbf{in}(j, S, t) \Leftrightarrow \mathbf{in}(j, S_1, t) \vee \mathbf{in}(j, S_2, t) \vee \ldots \vee \mathbf{in}(j, S_n, t)$

3. {S} **if** C **then** S_1 **else** S_2

   A8a. $\mathbf{at}(j, S, t) \wedge C(j, t) \Leftrightarrow \mathbf{at}(j, S_1, t)$
   A8b. $\mathbf{at}(j, S, t) \wedge \neg C(j, t) \Leftrightarrow \mathbf{at}(j, S_2, t)$
   A8c. $\mathbf{after}(j, S_1, t) \vee \mathbf{after}(j, S_2, t) \Leftrightarrow \mathbf{after}(j, S, t)$

4. {S} **while** C **do** S_1

   A9a. $\mathbf{at}(j, S, t) \wedge C(j, t) \Rightarrow \mathbf{at}(j, S_1, t)$
   A9b. $\mathbf{at}(j, S, t) \wedge \neg C(j, t) \Rightarrow \mathbf{after}(j, S, t)$
   A9c. $\mathbf{after}(j, S_1, t) \Rightarrow \mathbf{at}(j, S, t)$

5. {S} $x_1, x_2, \ldots, x_n := e_1, e_2, \ldots, e_n$ (multiple assignment)

   The multiple assignment statement updates a subset of local variables in parallel. It assigns e_1 to x_1, e_2 to x_2, ..., e_n to x_n in one atomic operation. We assume that all expressions

[2]To understand this, consider the following interpretation of **at** (j, S, t) and **after** (j, S, t): the timed formula at (j, S, t) is true if and only if at (j, S) is true for *some instant* when $\tau = t$; the timed formula **after** (j, S, t) is true if and only if **after** (j, S) is true for *some instant* when $\tau = t$. However, we cannot use this interpretation because, in our model, a fraction of a time unit cannot be measured.

on the right are evaluated first, and then the corresponding values are assigned to the variables on the left.

In the following axiom, Q (j) is a state predicate of station process j with no free occurrence of any of the WV and RO variables or the control location, and Q (j, t) is the timed formula defined from it.

Notation: $Q(j, t)\ [x_1, x_2, \ldots, x_n \mid e_1, e_2, \ldots, e_n]$ denotes formula Q (j, t) with each free occurrence of x_i replaced by e_i, for all i.

A10. $\mathbf{at}(j, S, t) \ \wedge\ Q(j, t)\ [x_1, x_2, \ldots, x_n \mid e_1, e_2, \ldots, e_n]$
$\Leftrightarrow$
$\mathbf{after}(j, S, t) \ \wedge\ Q(j, t)$

So far, we have stated the semantics of some Pascal-like constructs: assignment, if-then-else and while-do. It is assumed that the assignment statement does not take any time to execute, and only local variables can be updated using this statement. Similarly, the condition-test in the if-then-else and while-do constructs is assumed to take zero time. We will return to this point later.

Temporal Constructs

1. **set**

The **set** statement is a multiple-assignment statement that is used to update write variables of a station process in one atomic operation. It takes one time unit to execute.

{S} **set** $y_1, \ldots, y_m := e_1, \ldots, e_m$

where $y_1, \ldots, y_m$ are write variables (at least one of which is not an auxiliary variable). In the following, P (j) is a state predicate with no free occurrence of any RO variable or the control location, and P (j, t) is the timed formula defined from it.

A11. $\mathbf{at}(j, S, t) \ \wedge\ P(j, t)\ [y_1, \ldots, y_m \mid e_1, \ldots, e_m]$
$\Leftrightarrow$
$\mathbf{after}(j, S, t + 1) \ \wedge\ P(j, t + 1)$

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. Each boolean condition is a predicate over RO variables.

{S} **wait-seq** (pattern)

where

pattern $\equiv C_1$ **for** T_1; C_2 **for** T_2; ...; C_n **for** T_n.

The **wait-seq** statement causes a station 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 formula, match (j, pattern, t), defined below, is true iff station j observes a match of the pattern at time t, namely: 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. Let $C_i(j, t)$ denote the timed formula defined from $C_i(j)$.

$$\begin{aligned} \text{match}(j, \text{pattern}, t) \equiv\ & \langle \forall t' : t - T_n < t' \leq t : C_n(j, t') \rangle \wedge \\ & \langle \forall t' : t - (T_n + T_{n-1}) < t' \leq t - T_n : C_{n-1}(j, t') \rangle \wedge \\ & \vdots \\ & \langle \forall t' : t - (T_n + \ldots + T_1) < t' \leq t - (T_n + \ldots + T_2) : C_1(j, t') \rangle \end{aligned}$$

Execution of **wait-seq** terminates as soon as station j has observed the complete pattern. For this to happen at time t, the inputs (RO variables) should be such that match (j, pattern, t) is true *and* station j has been in **wait-seq** long enough to have observed those inputs. To specify this last condition, define

$$\begin{aligned} \text{control}(j, S, u, t) \equiv\ & (\textbf{at}(j, S, t-u) \vee \textbf{in}(j, S, t-u-1)) \wedge \\ & \langle \forall t' : t-u \leq t' < t : \textbf{in}(j, S, t') \rangle \wedge \\ & \langle \forall t' : t-u < t' < t : \neg\, \textbf{at}(j, S, t') \wedge \neg\, \textbf{after}(j, S, t') \rangle \end{aligned}$$

where u is a time parameter representing the duration of time for which control of station j has been in S.

The formula control (j, S, u, t) is true iff control of station j has been continuously in statement S throughout a time interval of length u (starting at time t − u). Control could have just reached S at time t − u, or could already be in S at time t − u − 1, and control has remained inside S continuously till t. If the inputs are such that the pattern begins at the same time as when control reaches **wait-seq**, the execution time for **wait-seq** is minimum and is equal to len (pattern) − 1, where len (pattern) = $T_1 + \ldots + T_n$.

The semantics of **wait-seq** can now be defined. In the following axioms, P (j) is a state predicate of station j with no free occurrence of any RO variable or the control location, and P (j, t) is the timed formula defined from it. Axiom A12a states that control is in **wait-seq** at time t if and only if control has just reached the statement at time t or was already in it at time t − 1, but the termination condition of the statement is not true (either the pattern has not been matched, or the statement has not been executing long enough). Axiom A12b states that execution of **wait-seq** will terminate as soon as, and only if, a match for the entire pattern is achieved. Axiom A12c states that if control is in **wait-seq** at time t, the write variables and local variables of the station process remain the same at time t + 1. Using A12c and the control axioms (A1-A5), we can derive the following result: if P (j) is true when control reaches **wait-seq**, then it remains true throughout the statement execution and upon its termination.

A12a. $(\textbf{at}(j, S, t) \vee \textbf{in}(j, S, t-1)) \wedge$
$(\neg\, \text{match}(j, \text{pattern}, t) \vee \neg\, \text{control}(j, S, \text{len}(\text{pattern}) - 1, t))$
$\Leftrightarrow$
$\textbf{in}(j, S, t)$

A12b. $\text{control}(j, S, \text{len}(\text{pattern}) - 1, t) \wedge \text{match}(j, \text{pattern}, t)$
$\Leftrightarrow$
$\textbf{after}(j, S, t)$

A12c. $\textbf{in}(j, S, t) \wedge P(j, t) \Rightarrow P(j, t+1)$

3. **wait-par**

The **wait-par** statement is an extension of the **wait-seq** statement. It allows a station to wait for several conditions in parallel (wait-par stands for wait-in-parallel).

Define

$\text{pattern}_i \equiv C_{i,1}$ **for** $T_{i,1}; \ldots; C_{i,n(i)}$ **for** $T_{i,n(i)}$

where n(i) is the number of conditions in pattern_i.

```
{S}    wait-par
              wait-seq (pattern_1) :: label_1
            || wait-seq (pattern_2) :: label_2
                 ⋮
            || wait-seq (pattern_m) :: label_m
        end-wait-par
```

The meaning of the **wait-par** statement is the following. When control reaches the **wait-par** statement, start 'executing' all **wait-seq** statements concurrently. (We will refer to each **wait-seq** as a *clause* of the **wait-par** statement.) The **wait-par** terminates as soon as any one of the clauses completes execution ('fires'). When that happens, control passes to the statement following **wait-par**, and all clauses inside **wait-par** are terminated. Associated with each clause is an identifying "label." The label associated with the clause that fired is *implicitly* assigned the value true, and all other labels are assigned the value false. The labels are used to select the piece of code to be executed after the **wait-par** statement.

The **wait-par** statement is kind of similar to Dijkstra's guarded command [2]. However, the "action" corresponding to each "guard" is stated after **wait-par**: the only action inside **wait-par** is the implicit assignment to labels. Also, unlike a guarded command, evaluation of guards is nonatomic, and control remains inside **wait-par** until one of the guards evaluates to true.

In the case of a guarded command, if two or more guards evaluate to true, a nondeterministic choice is made. However, nondeterministic behavior is not desirable for a real-time system. For this reason, we assign a priority to each clause in **wait-par**. If two or more clauses terminate successfully at the same time, only the label for the clause with the highest priority is assigned the value true. The clauses within **wait-par** are listed in decreasing order of priority, i.e., the clause with a lower index has a higher priority.

The axioms for **wait-par** are given below. These axioms are similar to those for **wait-seq**. Axiom A13a states that control remains within **wait-par** as long as, and only if, none of the patterns match. A13b states that **wait-par** terminates as soon as, and only if, inputs match (any) one of the patterns. Note that two or more patterns may achieve a match at the same time, but only the label with the lowest index is true when **wait-par** terminates. Below, timed formula $\text{label}_i(j, t)$ is true if and only if label_i is true for station j at time t.

A13a. $$(\textbf{at}(j, S, t) \lor \textbf{in}(j, S, t-1)) \land \langle \forall k :: \neg\, \text{match}(j, \text{pattern}_k, t) \lor \neg\, \text{control}(j, S, \text{len}(\text{pattern}_k) - 1, t) \rangle \Leftrightarrow \textbf{in}(j, S, t)$$

A13b. $\text{control}(j, S, \text{len}(\text{pattern}_i) - 1, t) \;\wedge\; \text{match}(j, \text{pattern}_i, t) \;\wedge$
$\langle \forall k : 1 \leq k < i :$
$\qquad\qquad \neg\, \text{match}(j, \text{pattern}_k, t) \;\vee\; \neg\, \text{control}(j, S, \text{len}(\text{pattern}_k) - 1, t) \rangle$
$\Leftrightarrow$
$\textbf{after}(j, S, t) \;\wedge\; \text{label}_i(j, t) \;\wedge\; \langle \forall k : k \neq i : \neg\, \text{label}_k(j, t) \rangle$

In addition to the above two axioms, Axiom A12c for **wait-seq** is also applicable to **wait-par**.

Special Cases of wait-seq

Two simple forms of the **wait-seq** statement are used frequently. They are given special names. Special cases of the **wait-seq** axioms are called rules. Apart from the rules given below, Axiom A12c is also applicable to these statements.

1. **delay** (T)
   $\equiv$ **wait-seq** (true **for** T + 1)

   The statement **delay** (T) causes a station to halt and remain idle for time T. It takes time T to execute. The rule for the **delay** statement is:

   A14. $\textbf{at}(j, S, t)$
   $\Leftrightarrow$
   $\langle \forall t' : t \leq t' < t + T : \textbf{in}(j, S, t') \rangle \;\wedge\; \textbf{after}(j, S, t + T)$

2. **wait** (C)
   $\equiv$ **wait-seq** (C **for** 1)

   where C is a predicate over RO variables only. The statement **wait** (C) causes a station 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, no time elapses in the execution of **wait** (C).) The time taken by the execution of **wait** (C) is indeterminate, as it depends upon when C becomes true. The rules for **wait** (C) are given below. C (j, t) denotes the timed formula defined from C (j).

   A15a. $(\textbf{at}(j, S, t) \;\vee\; \textbf{in}(j, S, t - 1)) \;\wedge\; \neg\, C(j, t)$
   $\Leftrightarrow$
   $\textbf{in}(j, S, t)$

   A15b. $(\textbf{at}(j, S, t) \;\vee\; \textbf{in}(j, S, t - 1)) \;\wedge\; C(j, t)$
   $\Leftrightarrow$
   $\textbf{after}(j, S, t)$

In a **wait-seq** statement, if the time interval 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).

Time for 'Internal' Computation

We have assumed that the only statements requiring non-zero execution time are the **set** and **wait** statements. Multiple-assignment statements and condition-testing for the if-then-else and while-do constructs are assumed to take negligible time. But this assumption may not always be valid. For example, the station process may need to execute an algorithm before updating a communication variable, and the algorithm may consist of several assignments to local variables. If the time required to execute the algorithm becomes appreciable, it has to be accounted for in the station program. This also applies to the testing of a complex condition, which may require a significant amount of time to evaluate. In these cases, a delay statement has to be introduced in the program to account for the computation time. The delay should be long enough for the computation to be completed. In general, such delays will depend upon the processor on which the program is run.

Channel Processes

For a given network, specification of the channel processes depends solely on the network configuration, and is independent of the protocol to be implemented by the station processes. We illustrate our model of channel processes by specifying the unidirectional bus system (UBS) configuration of Expressnet, which consists of a single cable folded as shown in Figure 1. (The bus is folded to achieve broadcast, i.e., so that a signal transmitted by a station can be received by all stations.)

The channel process specification consists of a description of the state of the channel, and some transition rules describing how the channel state changes with time. A bus is divided into segments, the length of a segment being the distance of signal propagation in one time unit. The segments are numbered 1 to N, and stations are numbered 1 to M, with the numbers increasing along the direction of signal propagation. The function loc (p) denotes the location of station p on the bus, i.e., the segment number of the cable where station p is connected. We assume that the first station is connected to the left-most segment, and the last station to the right-most segment, i.e., loc (1) = 1 and loc (M) = N.

The state of a 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 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 transition rules capture the propagation of signals, transmitted by one or more stations, along the channel.

In spite of its simplicity, this channel model is quite useful: it can model simultaneous transmissions by two or more stations, and can model the presence of several messages in the channel. A variety of broadcast networks with multiple buses can be specified; e.g., see [10].

The UBS configuration (figure 1) 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 **out-bus**, and that of the inbound channel by an array **in-bus**. The element **out-bus** (s) represents the status of the outbound bus at segment s, and **in-bus** (s) that of the inbound bus at segment s.

```
out-bus, in-bus : array [1..N] of Boolean
out-bus (s) : true, if a carrier is present on the outbound-bus at segment s
              false, otherwise
in-bus (s) : true, if a carrier is present on the inbound-bus at segment s
             false, otherwise
```

3 The Expressnet Protocol

In this section, we illustrate our model by specifying the Expressnet protocol [19]. First, we give an informal description of the Expressnet protocol in section 3.1. A formal specification follows in section 3.2.

3.1 Description of Expressnet

Expressnet belongs to a class of access protocols designed to exploit the directionality of signal propagation in a bus to provide collision-free, bounded-delay access to a shared bus. Several other configurations and access protocols have also been proposed for broadcast bus networks to achieve these goals [3, 4, 5, 19, 20]. Note that in broadcast bus networks, stations are connected to buses via taps. Since taps are passive elements, unlike repeaters in a ring which are active elements, these networks are somewhat less susceptible to node and link failures than rings. Conceptually, they employ a token-passing mechanism that is implicit and efficient, thus incorporating advantages of both bus and ring networks.

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. Each station also has the ability to sense the presence of 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 T_e. The propagation delay along the connector is T_c. The propagation delay between the outbound and inbound taps for each station is fixed and equal to $T_e + T_c$. The detection delay, i.e., the time taken by a station 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 an information *packet*. 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 that have packets to send transmit their TUs in a round-robin fashion. The succession of TUs transmitted in the same round is called a *train*.

Events EOC_{out} and EOT_{in}

Boolean variables $\mathbf{c_{in}}$ and $\mathbf{c_{out}}$ are used to indicate the presence of a carrier on the inbound and outbound channels, respectively. If $\mathbf{c_{in}}$ is true, the station *senses* a carrier on the inbound channel. Similarly, $\mathbf{c_{out}}$ indicates that the station senses a carrier on the outbound channel.

Next, we define channel events EOC_{out} (for end-of-carrier) and EOT_{in} (for end-of-train). An EOC_{out} event is said to occur when $\mathbf{c_{out}}$ changes from true to false. In defining the EOT_{in} event, we make use of the fact that consecutive TUs in a train are separated by a gap of $d + 1$ time units. Therefore, an EOT_{in} event occurs when $\mathbf{c_{in}}$ changes from true to false, and remains false for $d + 2$ time units.

Elements of the Access Mechanism

The basic access mechanism in the Expressnet protocol is the *attempt-and-defer* method.

Whenever a train is in progress on the outbound channel, each station that has a packet to send tries to transmit a TU as follows. Immediately following the detection of an EOC_{out} event, station p starts transmission of its TU. Simultaneously, it monitors the outbound channel (on the upstream side of its transmission tap) for the presence of a carrier. If another station, q, with index lower than that of p, has also started transmission after detecting the same EOC_{out} event, station p will detect a carrier within the first d seconds of its own transmission. If that happens, station p immediately aborts its current transmission, deferring to the one from the upstream station. Otherwise, it completes the transmission of its TU.

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

1. After all stations that have a packet to send have transmitted their TU's in the ongoing round, each station executes a procedure to start a new train. This consists of transmitting, following the detection of an EOT_{in} event, an unmodulated signal for duration d, called the *locomotive*. (By virtue of the Expressnet topology, locomotives transmitted by different stations overlap exactly in time.)
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** and $\mathbf{c}_{\mathbf{out}}$ with the outbound-bus process, and variable $\mathbf{c}_{\mathbf{in}}$ with the inbound-bus process. The inbound-bus and the outbound-bus processes share variable **out-bus** (N).

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

During its operation, station p continuously monitors the two buses, and takes actions based on the status of each bus. Therefore, $\mathbf{c}_{\mathbf{out}}$ and $\mathbf{c}_{\mathbf{in}}$ can be considered to be *inputs* to the station process (they are read-only variables).

While executing the access protocol, a station process transmits other signals besides the information packet itself. For clarity and ease of stating and verifying the desired property of collision-freedom, it is useful to distinguish the various signals transmitted by the station: pilot, locomotive, preamble, and packet. We introduce four auxiliary variables in the program to indicate the signal that is being transmitted. (Recall that auxiliary variables are used for specification and verification only, and do not have to be implemented.) These variables are: **tx-pilot, tx-loco, tx-preamble** and **tx-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.

Variables **packet-to-send**, **request-to-wake-up** and **request-to-sleep** are RO variables shared between the station process and a separate user process. Whenever **packet-to-send** is true, it indicates that the user process needs to transmit a packet. **request-to-wake-up** and **request-to-sleep** indicate that the user process wishes to wake up or sleep, respectively. The integer parameter **tx-delay** denotes the time used for a packet transmission.

Channel Events

Occurrences of various channel events are detected by conditions over time intervals.

1. To ascertain that an $\mathrm{EOC_{out}}$ event has occurred ($\mathbf{c_{out}}$ has changed from true to false), a station has to observe $\mathbf{c_{out}}$ to be true for one time unit and false for one time unit. Thus it detects an $\mathrm{EOC_{out}}$ event one time unit after end-of-carrier actually occurs.

2. Two consecutive TUs are separated by a gap of duration $\mathrm{d} + 1$ time units (sum of the delay to detect $\mathrm{EOC_{out}}$ and the detection delay). Event $\mathrm{EOT_{in}}$ actually occurs when $\mathbf{c_{in}}$ changes from true to false and there is a gap greater than $\mathrm{d} + 1$ in length on the inbound bus. But to detect the $\mathrm{EOT_{in}}$ event, a station has to observe $\mathbf{c_{in}}$ to be true for one time unit, and false for $\mathrm{d} + 2$ 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 $\mathrm{T_e} + \mathrm{T_c} + 2\mathrm{d} + 2$. A station detects the net to be "asleep" if does not observe a signal on the inbound bus for $\mathrm{T_e} + \mathrm{T_c} + 2\mathrm{d} + 3$ time units.

Station Program

The program for a station implementing the Expressnet protocol is given in Figure 2. The initial value of each Boolean variable is *false.*

4 Specification of Unidirectional Bus

In this section, we first introduce the notion of detection delay. Then, we state axioms for the channel processes of the UBS configuration, and some useful lemmas derived from them.

4.1 Detection Delay

The specification of a station process given in Section 3 makes use of the shared variables $\mathbf{c_{in}}$ and $\mathbf{c_{out}}$. These variables indicate the channel states as sensed by the station. But there is an inherent delay between the time a channel 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.

We define the following timed formulas for the inbound and outbound buses and for station j. These are used to characterize the detection delay.

out-bus (s, t) = true, if a signal is *actually* present at segment s of the outbound bus at time t
false, otherwise

in-bus (s, t) = true, if a signal is *actually* present at segment s of the inbound bus at time t
false, otherwise

$c_{in}(j, t)$ = true, if station j *senses* a carrier on the inbound channel at time t
false, otherwise

$c_{out}(j, t)$ = true, if station j *senses* a carrier on the outbound channel at time t
false, otherwise

The following axioms relate the actual and sensed signals :

CA.1 $c_{in}(j, t) \Leftrightarrow$ in-bus (loc (j), t − d)

CA.2 $c_{out}(j, t) \Leftrightarrow$ out-bus (loc (j), t − d)

4.2 Axioms for the Channel Processes

We first define the following timed formula for station j:

talk (j, t) = true, if station j is transmitting a signal at time t
false, otherwise

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

1. For the outbound bus :

CA.3 $\langle \forall s, t : 1 < s \leq N :$ out-bus (s, t) $\Leftrightarrow$ out-bus (s − 1, t − 1) $\vee \langle \exists\, i :$ loc (i) = s − 1 : talk (i, t − 1) $\rangle \rangle$

CA.4 $\langle \forall t ::$ out-bus (1, t) $\Leftrightarrow$ false $\rangle$

2. For the inbound bus :

CA.5 $\langle \forall s, t : 1 < s \leq N :$ in-bus (s, t) $\Leftrightarrow$ in-bus (s − 1, t − 1) $\rangle$

CA.6 $\langle \forall t ::$ in-bus (1, t) $\Leftrightarrow$ (out-bus (N, t − T_c) $\vee$ talk (M, t − T_c)) $\rangle$

4.3 Basic Lemmas for Channel Processes

Using the axioms stated above, the following lemmas can be proved [11]. In the following, $pdelay_{ij}$ denotes the propagation delay from the transmit port of station i to the transmit port of j. For Expressnet, it is the same as the propagation delay between their receive ports. Since the bus is unidirectional, it is defined only if i is upstream of j, that is, $i < j$.

Lemma C.1 states that any two stations observe the same sequence of events on the inbound channel, with a delay equal to the propagation delay between them.

__Lemma C.1__ $\langle \forall i : i < j : c_{in}(j, t) \Leftrightarrow c_{in}(i, t - pdelay_{ij}) \rangle \wedge$
$\langle \forall k : k > j : c_{in}(j, t) \Leftrightarrow c_{in}(k, t + pdelay_{jk}) \rangle$

For notational convenience, we define another term,

$$delay_{ij} = \begin{cases} pdelay_{ij}, & \text{if } i < j \\ -1 \times pdelay_{ij}, & \text{if } i > j \end{cases}$$

From now on, we will use $delay_{ij}$ instead of $pdelay_{ij}$. With this notation, Lemma C.1 can be restated as follows:

$$c_{in}(i, t) \Leftrightarrow c_{in}(j, t + delay_{ij})$$

Lemmas CMR.1 to CMR.4 state timing relations between events at different points along a channel and their effects at remote points. These lemmas, called *communication rules*, are used in our verification method described below (see Section 5.3).

Lemmas CMR.1 and CMR.2 relate the instant when a station transmits a signal to the time when a signal is detected on the outbound channel.

Lemma CMR.1 $talk(i, t) \Rightarrow \langle \forall j : i < j \leq M : c_{out}(j, t + delay_{ij} + d) \rangle$

Lemma CMR.2 $\langle \forall i : 1 \leq i < j : \neg\, talk(i, t - delay_{ij} - d) \rangle \Rightarrow \neg\, c_{out}(j, t)$

Lemmas CMR.3 and CMR.4 are analogous to Lemmas CMR.1 and CMR.2, and state similar timing relations for the inbound channel.

Lemma CMR.3 $talk(i, t) \Rightarrow \langle \forall j :: c_{in}(j, t + T_e + T_c + d + delay_{ij}) \rangle$

Lemma CMR.4 $\langle \forall i :: \neg\, talk(i, t - (T_e + T_c + d + delay_{ij})) \rangle \Rightarrow \neg\, c_{in}(j, t)$

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 detected by each of the other stations as a change in the state of a bus. The detection is made at another station after a time delay that is equal to the propagation delay between the two stations plus the detection delay, d. Our model captures these timing relations.

5 Verification Method

In Section 5.1, we present two types of protocol properties of interest to us in this paper. For a given protocol, our proof technique is based upon the construction of a *timed reachability graph* (TRG) for the protocol. Sections 5.2 to 5.4 are concerned with the representation and construction of TRGs. Techniques to reduce the size of an actual TRG representation are described in Section 5.5. In Section 5.6, we describe how to prove protocol properties using a TRG.

5.1 Types of System Properties

In this section, we give the general form of two types of protocol properties of interest to us. In the following, P (i, t) and Q (j, t) denote interval formulas of station processes i and j, respectively. The total number of station processes is denoted by M.

Safety Properties

One type of properties we use is:

$$P(i, t) \Rightarrow Q(j, t + L_{ij})$$

where, for each pair of station processes, i and j, L_{ij} is an integer constant (possibly negative) that depends only upon the topology of the system. For example, L_{ij} is equal to the propagation delay between stations i and j in Expressnet. Note that the above property relates the states of station processes at different time instants that are dependent on the network topology. It imposes a restriction on station process states, and we refer to it as a safety property.

Progress Properties

The following property of *progress within bounded delay* is also used:

$$P(i, t_1) \Rightarrow \langle \exists t_2 : 0 \leq t_2 - t_1 \leq \Delta : Q(j, t_2) \rangle$$

where Δ is a constant denoting some bound. The property states that whenever interval formula P is true for station i, interval formula Q will become true for station j within a bounded delay. In the special case of $i = j$, the property states that station i makes progress within a bounded delay (this is stronger than saying "process i eventually makes progress" using temporal logic).

5.2 Local and Global Formulas

The state of station process i is described by a *local formula* that is the conjunction of the following set of formulas:

- a *station formula* SF (i, t) describing the state of the station process over a time interval. Specifically, SF (i, t) is a conjunction of interval formulas, each of which is defined from a state predicate with no free occurrence of any RO variable shared with a channel process.
- a *channel formula* CF (i, t), for every RO variable shared between station process i and a channel process, describing the value of the shared RO variable over a time interval. Specifically, CF (i, t) is a conjunction of interval formulas, each of which is defined from a predicate on the shared RO variable and global clock variable τ.

For station process i, let SF (i, t) denote its station formula, and $\{CF_1(i, t), \ldots, CF_n(i, t)\}$ denote its set of channel formulas. The local formula describing station process i is defined to be

$$LF(i, t) \equiv SF(i, t) \wedge CF_1(i, t) \wedge \ldots \wedge CF_n(i, t)$$

Convention: If the station formula or a channel formula is missing in the definition of a local formula, the missing formula is assumed to be *true*.

A *global formula* is defined to be the conjunction of the local formulas for all station processes (indexed by 1, ... , M), namely:

$$GF_1(t) \equiv LF(1, t) \wedge \ldots \wedge LF(M, t)$$

where the time parameter t, the same for each local formula, is a free variable. Conceptually, a global formula represents a description of the states of all station processes in the network. In fact, interval formulas within the global formula provide descriptions of subsets of state variables over different time intervals.

Let $GF_1(t), \ldots, GF_k(t)$ be global formulas. Then, the conjunction $GF_1(t) \wedge \ldots \wedge GF_k(t)$ is also a global formula.

Notation: We will use an optional subscript to distinguish between different station formulas of the same process. Thus SF (i, t) denotes some station formula of process i, while $SF_j(i, t)$ and $SF_k(i, t)$ denote two different station formulas of process i. The same convention will be used for channel formulas as well.

5.3 Transition Rules

Each node in the TRG for a real-time protocol is a global formula. Starting from a global formula describing the initial condition of the protocol, the TRG is constructed using two types of transition rules, *communication rules*, and *control flow rules*. The application of a transition rule to derive one global formula from another corresponds to an arc in the TRG. In what follows, we first introduce the two types of rules, and then illustrate their applications to some examples.

Communication Rules

Communication rules are stated in the following general form:

$$\textbf{CMR} \qquad \mathrm{SF}(1, u - L_{1j}) \wedge \mathrm{SF}(2, u - L_{2j}) \wedge \ldots \wedge \mathrm{SF}(M, u - L_{Mj}) \Rightarrow \mathrm{CF}(j, u)$$

where u is a free-occurring time parameter, and some of the station formulas in CMR may be *true*. For Expressnet, the communication rules are precisely the channel lemmas CMR.1-CMR.4 stated in Section 4.3. Lemmas CMR.1 and CMR.3 model the transmission of a signal by station i that is subsequently detected by station j after a propagation delay of L_{ij} (different values for inbound and outbound channels). Lemmas CMR.2 and CMR.4 model the combined effect of *multiple* stations on another station: for instance, CMR.4 states that if all stations are not transmitting a signal, then no signal will be detected on the inbound bus.

Control Flow Rules

The sequential program specifying the behavior of a station process is transformed into a set of control flow rules. To obtain these rules, the program is first analyzed to identify *input locations* in the program. Input locations are defined to be those places in a station program where RO variables are read. Thus, input locations are precisely those places where the program behavior may be affected by inputs from the station's environment (its user process, and channel states as sensed by the station).

Input locations can be found in the following:

- **wait-seq** and **wait-par** statements
- multiple-assignment and **set** statements
- condition tests within **if-then-else** and **while-do** statements

For input locations found in **wait-seq** and **wait-par** statements, control flow rules are obtained from Axioms A12a and A12b, and A13a and A13b. (These rules may be refined by applying Axiom A12c.) For input locations found in multiple-assignment and **set** statements, control flow rules are obtained from Axioms A10 and A11.

Having identified input locations in a station program, the axioms of Pascal-like constructs are applied to the program to identify the flow of control from one input location to another. The behavior of the station program between two input locations is characterized by a control flow rule (derived as described below). The rule specifies how the state of the station process evolves over a time interval. Specifically, consider the *logical* sequence of statements S_{m+1}, S_{m+2}, ..., S_{n-1} between two input locations in statements S_m and S_n shown below. Suppose there is no input location in statements S_{m+1}, S_{m+2}, ..., S_{n-1}, which are treated as a *composite statement*.

```
                   ⋮
{S_{m-1}}    statement_{m-1};
{S_m}        statement_m;
                                {(τ = t_1) ∧ POST_m(j, t_0)}
{S_{m+1}}    statement_{m+1};
                                {(τ = t_1 + Δ_{m+1}) ∧ POST_{m+1}(j, t_0)}
{S_{m+2}}    statement_{m+2};
                   ⋮
{S_{n-1}}    statement_{n-1};
                                {(τ = t_1 + Δ_{m+1} + ...+ Δ_{n-1}) ∧ POST_{n-1}(j, t_0)}
{S_n}        statement_n;
{S_{n+1}}    statement_{n+1};
                   ⋮
```

In the above, $POST_i(j, t_0)$ is an interval formula that describes the state of the station during the execution of statement$_i$, and Δ_i is the time taken to execute statement$_i$; $POST_i(j, t_0)$ and Δ_i are obtained from the axioms in Section 2 defining statement$_i$. The following control flow rule is derived for the composite statement:

after $(j, S_m, t_1) \wedge POST_m(j, t_0)$
$\Rightarrow$
at $(j, S_n, t_1 + \Delta_{m+1} + \ldots + \Delta_{n-1}) \wedge POST_{m+1}(j, t_0) \wedge \ldots \wedge POST_{n-1}(j, t_0)$

A single control flow rule is specified for the composite statement because between S_m and S_n, the flow of control depends entirely upon the station's local and write variables.

For example, in the Expressnet station program given in Figure 2, the statements labeled $S_1, \ldots, S_7$ have input locations. In fact, for the sake of clarity, only statements having input locations are labeled (once input locations and control flow rules have been obtained, labels of other statements can be removed).

Control flow rules have the following general form:

CFR $\quad SF_j(i, u) \wedge F(i, u) \Rightarrow SF_k(i, u)$

where u is a free-occurring time parameter, $F(i, u)$ is a formula constructed from interval formulas of station i, and $SF_j(i, u)$ and $SF_k(i, u)$ are station formulas of station i that satisfy a *time continuity* property, i.e., the beginning of the time interval in $SF_k(i, u)$ is before, at or immediately following the end of the time interval in $SF_j(i, u)$. Note that the conjunction of $SF_j(i, u)$ and $SF_k(i, u)$ provides a description of the station state over the union of the two time intervals.

Informally, a control flow rule specifies the following: the "internal" state of station process i over a time interval, together with its inputs over various time intervals, determine the internal state of station process i over the "next" time interval. Note that each station program is deterministic and sequential. Therefore, during program execution, the behavior of a station over the next time interval is specified by *precisely one* control flow rule.

For the Expressnet station program in Figure 2, thirty control flow rules are obtained (see Appendix A).

5.4 Timed Reachability Graph

For a given protocol, the communication rules and control flow rules are first specified. The initial condition of all station processes is described using an *initial* global formula.

Consider a global formula, namely,

$$GF_1(t) \equiv LF(1,t) \wedge LF(2,t) \wedge \ldots \wedge LF(M,t)$$

A control flow rule of the form

$$SF_j(i,u) \wedge F(i,u) \Rightarrow SF_k(i,u)$$

is said to be *enabled* by $GF_1(t)$ if and only if there exists a substitution for every free time parameter in $SF_j(i,u) \wedge F(i,u)$ such that

$$LF(i,t) \Rightarrow (SF_j(i,u) \wedge F(i,u))[u \mid e(t)]$$

where the expression $e(t)$ is substituted for u. (For notational convenience, it is assumed that u is the only free time parameter in $SF_j(i,u) \wedge F(i,u)$. Some control flow rules have a second free time parameter.) If a control flow rule is enabled by $GF_1(t)$, application of the rule results in a new station formula, namely, $SF_k(i,u)\,[u \mid e(t)]$. A new global formula is obtained either by conjoining the new station formula to $LF(i,t)$, or using it to replace the old station formula in $LF(i,t)$.

When a control flow rule of station i is not enabled by $GF_1(t)$, it is sometimes because some channel formulas specified in the antecedent of the control flow rule are missing in the local formula $LF(i,t)$ within $GF_1(t)$. To supply the missing channel formulas, we apply communication rules to $GF_1(t)$ and possibly other global formulas along the path from the initial global formula to $GF_1(t)$. Application of a communication rule also requires a substitution of any free time parameter in the rule as described above (but the time continuity property need not be satisfied).

Observe that the only source of nondeterminism in our protocol model is input from user processes. (This is because user processes are not explicitly modeled.) For a given global formula, if a station is at a control location where there is no user input, there is exactly one control flow rule that determines the future behavior of the station. On the other hand, if the station is at a control location where user input is possible, then multiple control flow rules are applied, one for each possible value of the user input. Each rule application, if it is enabled, results in a new station formula and thus a new global formula, and corresponds to an arc in the TRG.

In Expressnet, the shared variables for user input are: **request-to-wake-up** (in statement S_1), **request-to-sleep** (in statement S_4), and **packet-to-send** (in statement S_6). To illustrate the application of control flow rules, consider the following global formula for the Expressnet protocol. It is labeled as **F1** in Appendix B.

F1: $\langle \forall j :: \mathbf{in}(j, S_1, t_0 - 1) \wedge \neg\, \mathrm{talk}(j, t_0 - 1) \wedge$
$\qquad \langle \forall t' : t_0 - 1 \le t' < t_0 + T_d : \neg\, c_{in}(j, t') \rangle \rangle$

We now show some of the intermediate steps in the derivation of formula **F5** in Appendix B. In global formula **F1**, the local formulas for all stations are the same. The control in each station is in an input location (S_1). Consider the local formula for station k. There are two possible input values, *true* and *false* for the RO variable request-to-wake-up. The station formula for k in **F1** can be written as:

f1: $\mathbf{in}(k, S_1, t_0 - 1) \wedge \neg \text{talk}(k, t_0 - 1) \wedge$
$(\text{request-to-wake-up}(k, t_0) \vee \neg \text{request-to-wake-up}(k, t_0))$

For the input value *false*, applying control flow rule CFR.3 (in Appendix A) to station formula *f1*, we get the following new station formula:

f2: $\mathbf{in}(k, S_1, t_0) \wedge \neg \text{talk}(k, t_0)$

For the input value *true*, applying control flow rule CFR.4 (in Appendix A) to station formula *f1*, we get the following new station formula:

f3: $\mathbf{after}(k, S_1, t_0) \wedge \neg \text{talk}(k, t_0)$

In applying each rule, we substitute $(t_0 - 1)$ for the time parameter u in the control flow rule to obtain the new station formula.

Therefore, the formula describing station k in the formula that represents **F1**'s successor in the TRG can be written as:

f4: $(\mathbf{in}(k, S_1, t_0) \vee \mathbf{after}(k, S_1, t_0)) \wedge \neg \text{talk}(k, t_0)$

The same rules can be applied to all stations, and we obtain the following new formula from **F1**:

f5: $\langle \forall j :: (\mathbf{in}(j, S_1, t_0) \vee \mathbf{after}(j, S_1, t_0)) \wedge \neg \text{talk}(j, t_0) \wedge$
$\langle \forall t' : t_0 - 1 \le t' < t_0 + T_d : \neg c_{in}(j, t') \rangle \rangle$

Note that the above formula is actually a disjunction of global formulas, and it represents a set of nodes in the TRG.

Termination of TRG

Let m and n denote two nodes in the TRG. Node n is said to be a *repeat* of node m if they are specified by global formulas $GF_n(t)$ and $GF_m(t)$, respectively, that satisfy the following conditions:

T.1 $GF_m(t)$ contains sufficient information such that no descendent of node m in the TRG is derived from information contained in an ancestor of node m.

T.2 For every station, there is precisely one control flow rule enabled by $GF_m(t)$.

T.3 $GF_n(t) \Rightarrow GF_m(t)[t \mid t + \Delta]$, for some constant Δ

where t is a free time parameter, and Δ represents a time shift between the two nodes.

If node n is a repeat of node m, the protocol behavior following node n is the same as the protocol behavior following node m, except for the absolute time; this is because transition rules are independent of the absolute time. Therefore, we can terminate a path when a repeat node is generated. TRG construction for a given protocol terminates when every path is terminated.

In Appendix B, we provide a representation of the TRG constructed for Expressnet (as modified in the following section). In general, there is no guarantee that TRG construction terminates for a given protocol.

5.5 Reduction Techniques for Representing TRG

Suppose the TRG of a given protocol is finite. However, if each rule application is represented as a distinct arc in a graph, the graph would be extremely large. To reduce the size of an actual TRG representation, various techniques can be employed. These techniques fall into four categories.

Aggregation of Nodes

We can represent a set of nodes (global formulas) by a single aggregate node. The formula of the aggregate node is a disjunction of the global formulas in the set. In particular, we use existential quantification over the station index in a global formula to represent an aggregate node. There are many examples of such formulas in Appendix B.

Concurrent Application of Sequences of Control Flow Rules

Suppose node m follows node n in the TRG. The formula of node m may be derived from the formula of node n by applying a sequence of control flow rules to station i, for all i. For each station, we can proceed with control flow rule application as long as one such rule is enabled (keeping in mind that branching occurs at any control location where there is user input). For each station, we stop if no control flow rule is enabled. If we have stopped for all stations, then we apply communication rules to strengthen the global formulas of node m so that control flow rules become enabled for some stations.

Partitioning an Aggregate Node

When a new channel formula needs to be derived, some communication rules are applied to global formulas in an aggregate node. In TRG construction, we may generate an aggregate node such that a particular communication rule is not enabled for all global formulas in the aggregate node. When this happens, it is convenient to partition the aggregate node into smaller (aggregate) nodes, such that the communication rule becomes enabled for the set of global formulas in one of the partitions.

For example, node 9 in the TRG is partitioned into nodes 10 and 11. Node 10 corresponds to the case in which no station is ready, and none will transmit a signal. Node 11 corresponds to the case in which at least one station is ready, and all ready stations will transmit a preamble. Rule CMR.4 is applied to node 10 as part of the derivation of node 12. Rule CMR.3 is applied to node 11 as part of the derivation of node 25.

We may also partition an aggregate node into smaller successor nodes for other reasons. In particular, we partition an aggregate node if one or more of the successor nodes leads to a repeat node. For example, node 13 is partitioned into node 15 and node 16. Node 15 leads to node 17, which repeats node 7, and node 16 leads to node 18, which repeats node 1.

Use of Induction in TRG Construction

In constructing the TRG of the modified Expressnet protocol for M stations, where M is a parameter, we make use of induction in the following manner. Observe that in each round of transmissions, the protocol behavior is the same for each transmission except for the index of the transmitting station. We exploit this fact to reduce the size of TRG representation. Instead of generating an entire path until a repeat node is obtained (in fact, this is not doable if M is a parameter), we "factor" out the repetitive part and treat it as a separate subgraph. The actual TRG contains multiple instances of this subgraph, one for each value of station index.

In the TRG for modified Expressnet, nodes 26 through 33 represent a subgraph which describes the protocol behavior during packet transmission by station m, where m is the index of the transmitting station.

5.6 Proving properties

Recall that each application of a control flow rule results in a new station formula with the time continuity property . Because of the time continuity property, the conjunction of all global formulas along a path in the TRG provides a description of the state of every station in the network over continuous time. Protocol properties are proved by showing that they are invariant over time for every path in the TRG.

For example, consider a safety property of the form

$$P(i, t) \Rightarrow Q(j, t + L_{ij})$$

where $P(i, t)$ and $Q(j, t + L_{ij})$ are timed formulas. To prove the safety property for given values of i and j, we consider every path of the TRG. Let $F_{path}(\tau)$ denote the conjunction of all global formulas in a given path, where the global clock variable τ is shown explicitly. For time value t', we use $F_{path}(t')$ to denote the formula derived from $F_{path}(\tau)$ by assigning the value t' to τ.

To prove the above safety property, we check that one of the following holds for all $t' \geq t_0$, where t_0 is the starting time of protocol execution:

- $F_{path}(t') \Rightarrow \neg P(i, t')$
- $F_{path}(t') \Rightarrow P(i, t')$ and $F_{path}(t' + L_{ij}) \Rightarrow Q(j, t' + L_{ij})$

6 Expressnet Properties

First, we state the desired properties of Expressnet. Next, we show that the original protocol is susceptible to collisions, and propose a modification to it. Finally, we prove that the modified protocol is collision-free and derive a bound on its access delay.

6.1 System Properties

For Expressnet, the safety property of interest is freedom from collision, i.e., the *packet* portion of a station's transmission does not overlap with any other transmission. The progress property of interest is that each station gets a chance to transmit a packet within a bounded delay.

Notation: For every boolean variable v of station j, the timed formula defined from it is denoted by v (j, t).

Collision Freedom

The property of collision-freedom is stated below as **P.1**. The predicate tx-packet (i) is true if station i is transmitting a packet, and predicate talk (i) is true if station i is transmitting any signal.

P.1 $\text{tx-packet}(i, t) \Rightarrow \langle \forall j : i \neq j : \neg\, \text{talk}(j, t + \text{delay}_{ij}) \rangle$

If station i transmits a packet at time t, the signal will reach the transmit port of station j, downstream of it, at time $t + \text{delay}_{ij}$. Therefore, station j should not transmit any signal at time $t + \text{delay}_{ij}$. If a station, k, upstream of i transmits a signal at time $t - \text{pdelay}_{ki}$, the signal will reach i's transmission port at time t. Therefore, k should not transmit a signal at time $t + \text{delay}_{ik}$ (recall that $\text{delay}_{ik} = -\,\text{pdelay}_{ki}$).

Bounded Delay

A station that has a packet to send can be guaranteed to transmit the packet only if it remains awake long enough to do so. Thus the bounded-delay property should be stated as a conditional property. We first define $W(i, t_1)$, which represents a 'wakefulness' condition: if station i has a packet to send at time t_1, it will not be asked by its user process to go to sleep unless it has finished transmitting that packet.

Timed formula $\text{pts}(i, t)$ is true iff the variable **packet-to-send** is true for station i, and the interval formula $\text{packet-sent}(i, t)$ is true if station i completes transmission of its packet at time t. Below, P_i is a parameter denoting the time taken by station i to transmit a packet .

$$\text{packet-sent}(i, t) \equiv \langle \forall t' : t - P_i < t' \leq t : \text{tx-packet}(i, t') \rangle$$

$$\begin{aligned} W(i, t_1) \equiv \langle \forall t_2 : t_2 > t_1 : \; & \text{pts}(i, t_1) \wedge \langle \forall t' : t_1 \leq t' \leq t_2 : \neg\, \text{packet-sent}(i, t') \rangle \\ & \Rightarrow \\ & \langle \forall t' : t_1 \leq t' \leq t_2 : \neg\, \text{request-to-sleep}(i, t') \rangle \rangle \end{aligned}$$

The bounded-delay property is stated below.

P.2 $\text{pts}(i, t_1) \wedge W(i, t_1) \Rightarrow \langle \exists t_2 : 0 \leq t_2 - t_1 \leq \text{Bound} : \text{packet-sent}(i, t_2) \rangle$

That is, if station i has a packet to send at time t, and it satisfies the 'wakefulness' condition, then it will transmit its packet within a bounded delay given by 'Bound'.

6.2 Scenario for Collision in the Original Protocol

The Expressnet protocol as specified in [19] does not guarantee collision-free access to the bus. Consider the following scenario. Stations j, r and s ($1 \leq j < r < s \leq M$) are awake and have finished transmitting their packets in the ongoing round. In time, each of them detects an EOT_{in} event and transmits a locomotive of length d units. Assume that stations 1 through r do not have a packet to transmit by the time they finish transmitting their locomotives. They will stop transmission, and will start waiting for the next EOT_{in} or EOC_{out} event. If station s has a packet to send immediately following the locomotive, it proceeds to transmit its preamble. At the end of the preamble, s 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 detection delay, r will detect the end of j's locomotive (incorrectly) as an EOC_{out} event d time units after it completes its own locomotive transmission. If by then it has received a packet to transmit, station r will start transmitting its preamble and packet, which will collide with the packet transmission of station s. This violates the desired property of collision-freedom. The above scenario is formally stated and proved as a theorem in [11].

6.3 Modified Expressnet Protocol

We now propose a modification to the Expressnet protocol to make it collision-free.

Modification

In order to rectify the problem, it is sufficient that the end of locomotive is not confused with the end of a packet. Since the $\mathrm{EOC_{out}}$ event itself cannot be prevented from occurring, we ensure that a station 'does not observe' that event. To this end, we propose the modification below.

Following a 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. Note that if a station receives a packet to transmit after it has had a chance to transmit in the current round, the packet will have to wait till the next round.

In fact, the above modification prevents only *active* stations from detecting the end of locomotive. (A station is said to be active if it is executing the while loop labeled S_4, which corresponds to a "regular round" of the protocol; otherwise, it is said to be *inactive*.) A station, which was inactive when $\mathrm{EOT_{in}}$ occurred, could become active before the locomotive has gone past its transmit port, and detect the end of locomotive as an $\mathrm{EOC_{out}}$ event. But as shown in [11], this cannot happen if the network parameters satisfy the condition: $T_e + T_c \geq d$. Since detection delay is typically much smaller than the end-to-end bus propagation delays, we expect this condition to be true. Henceforth, we assume that this condition is satisfied by the network.

Remark: Our modification does not increase the delay incurred by any packet at any station (except for those packets which, if transmitted, would cause a collision). This is because the time spent in this wait is less than the delay until the next $\mathrm{EOT_{in}}$ or $\mathrm{EOC_{out}}$ event.

Modified Station Program

The only change in the program is in the **then** clause of statement S_6. The rest of the program is the same as given in Figure 2. We give in Figure 3 the segment of the station program corresponding to the regular round only. The cold-start procedure is the same as that for the original protocol in Figure 2.

6.4 Proofs of Properties for the Modified Protocol

The modified Expressnet protocol is verified using the method described in Section 5.

Description of the TRG

Using the method described in Sections 5.4 and 5.5, a timed reachability graph (TRG) is constructed for the modified Expressnet protocol (see Appendix and Figure 4). The derivation of each (aggregate) node in the graph is by proving a lemma. (See [11] for lemmas and their proofs.)

We take advantage of the fact that all station programs in Expressnet are identical to reduce the size of the graph; specifically, a set of nodes are aggregated into a single node by existentially quantifying over station index in a global formula (nodes 3, 5-11, 14, 15, 17, 20-22, 24, 25, 31 and 33 of TRG in Appendix).

Time parameter t_0 represents the starting time of protocol execution. Node 1 represents the dormant state of the network: every station is asleep at time $t_0 - 1$ and the inbound channel is idle for the interval from $t_0 - 1$ to $t_0 + T_d$. If no station receives a request to wake up from its user process (represented by node 2), the network will remain dormant at time t_0 (node 4). If some station receives a request (node 3), it will wake up at time t_0 (node 5). Nodes 6 through 9 describe the protocol behavior during cold start.

Following the end of pilot-transmission (node 9), if there is a ready station following pilot-transmission (node 11), a train will start immediately following the pilot, with a gap of $d+1$ between the pilot and the first packet. Node 25 represents the condition for the left-most ready station to transmit a packet successfully. On the other hand, if none of the stations is ready to send a packet (node 10), the network is in the state prior to the start of a regular round (node 12).

If in the state prior to the start of a regular round some station is awake (node 14), a regular round will start (node 19) and all active stations will transmit a locomotive (node 20). If all stations are asleep (node 13), then either the network will return to the dormant state (node 18), or some station will wake up and undertake cold-start (node 17).

If there is a ready station following locomotive-transmission (node 22), the left-most ready station will satisfy the condition for transmitting a packet (node 24). If no station is ready following a locomotive (node 21), the 'train' is empty and the network will return to the state prior to the start of a regular round (node 23).

Nodes 26 through 33 constitute an induction step in the TRG construction. Node 26 represents the induction hypothesis that a station, say m, satisfies the condition for packet transmission. It will transmit a packet without collision (nodes 27 and 28). If at the end of packet transmission (node 29), there are some ready stations downstream (node 31), then the left-most of these stations will satisfy the condition for packet transmission (node 33). If at the end of a packet transmission by station m, no station downstream of m is ready to transmit (node 30), the network will return to the state at the start of a regular round (node 32).

The formula of node 24 (also node 25 and node 33) *implies* the formula of node 26. (We use a solid line and an arrow in Figure 4 to indicate "implication".)

Each repeat node is labeled by an asterisk (*) and a dashed line is used to indicate the node being repeated. Specifically, node 4 repeats node 1 with a time shift of one unit. Node 32 repeats node 12 (start of a new train): the time shift is the length of a train plus the inter-train gap. Node 23 also repeats node 12: the time shift is the length of a locomotive plus the inter-train gap. Node 18 repeats node 1 (dormant state), and node 17 repeats node 7 (start of pilot transmission).

Collision Freedom

The property of collision freedom (stated in section 6.1) is proved by examining each node in the TRG, as discussed in Section 5.6. To determine if the state predicate in a station interval formula of process i implies each of the predicates: tx-packet (i), ¬ tx-packet (i) or talk (i), we make use of the following invariants for a station program. These invariants can be easily proved.

PI.1 talk (i, t) $\Leftrightarrow$ tx-pilot (i, t) $\vee$ tx-loco (i, t) $\vee$ tx-preamble (i, t) $\vee$ tx-packet (i, t)

PI.2 $\text{tx-pilot}(i, t) \vee \text{tx-loco}(i, t) \vee \text{tx-preamble}(i, t) \Rightarrow \neg\, \text{tx-packet}(i, t)$

Using these invariants, we see that for each node in the TRG, except for node 27, tx-packet (i) is false for the entire interval of the station formula of process i in the node, for every i. Therefore, the collision-freedom property is trivially satisfied for these nodes.

In node 27, the following interval formula is part of the station formula of station m. Let P_m be a constant denoting the time taken by station m to transmit a packet:

$$\langle \forall t' : t_5 + \text{delay}_{1m} + d + 2 \ \leq t' \leq t_5 + \text{delay}_{1m} + P_m + d + 1 : \text{tx-packet}(m, t') \rangle$$

In node 28, which follows node 27 in a TRG path, the following interval formula is part of the station formula of each station k ($k \neq m$):

$$\langle \forall t' : t_5 + \text{delay}_{1k} + d + 2 \ \leq t' \leq t_5 + \text{delay}_{1k} + P_m + d + 1 : \neg\, \text{talk}(k, t') \rangle$$

Thus, for every value of t' when station m is transmitting a packet, every other station, k, is not transmitting a signal at time $t' + \text{delay}_{mk}$.

Therefore, the collision-freedom property is satisfied along every path of the TRG for all time.

Bounded Delay

To calculate the maximum delay from the time a station receives a packet to the time of successful transmission of the packet, we distinguish two cases: (1) delay of the first packet after the station wakes up, and (2) delays of subsequent packets.

The bounded delay property is proved by examining all paths in the timed reachability graph. Since **packet-to-send** can become true at any time, we have to consider the longest path from any node to the node in which the station transmits a packet (node 27).

Delay of the first packet

The worst-case delay for the first packet is when the station receives the packet at the same time it receives the request to wake up, and the net is asleep (node 5). We consider both paths from node 5 to node 27. Nodes 5-9 are common to both paths and correspond to station i waking up, detecting the net to be asleep, and transmitting a pilot.

Following node 9, one path to node 27 is via node 10. In this case, no station downstream of station i is ready to transmit a packet immediately after pilot transmission. Station i waits for a regular round to start (node 19) and gets a chance to transmit its packet after all stations upstream of it have had a chance to transmit. For station i, the worst-case delay for the first packet along this path is given by

$$\text{Bound}_1(i) = i \times (P_{max} + 2d + 2) + 3 \times (T_e + T_c) + 5d + 4$$

where P_{max} is the maximum packet-transmission time over all stations. $\text{Bound}_1(i)$ is largest for $i = M$.

The second path from node 9 to node 27 is via node 11. In this case, station i waits for all stations downstream of it to transmit their packets, and for a new round to start (node 19). Station i gets a chance to transmit in the second round after all stations upstream of it have had a chance to transmit. The worst-case delay along this path is given by

$$\text{Bound}_2 = \text{M} \times (\text{P}_{\text{max}} + 2\text{d} + 2) + 3 \times (\text{T}_\text{e} + \text{T}_\text{c}) + 5\text{d} + 4$$

Bound_2 is independent of the station index, and is the same as Bound_1 (i) for i = M.

Delay of a subsequent packet

The worst-case delay is when a station receives a packet to transmit as soon as it has finished transmitting the previous packet (node 27). Since a station can transmit only one packet in every round, it has to wait until the current round is over (node 32) and the next round starts (node 19), before it gets a turn to transmit again (node 27). In this case, the maximum delay is given by

$$\text{Bound}_3 = \text{M} \times (\text{P}_{\text{max}} + 2\text{d} + 2) + \text{T}_\text{e} + \text{T}_\text{c} + 2\text{d}$$

Note that the worst-case delay for subsequent packets is the same for all stations.

Remark: Properties of the modified Expressnet protocol are proved under very general assumptions. We have made no assumption as to which stations are awake or asleep, the time when a station wakes up or goes to sleep, or which stations have packets to send in any given round.

7 Conclusions

As illustrated by our analysis of the Expressnet protocol, timing bugs are hard to catch. To improve our confidence in the implementation of a real-time protocol, a careful analysis of time-dependent interactions between station programs should be carried out using a formal method. In this paper, we have presented a specification model and a formal analysis method.

Our model and method have certain desirable characteristics. First, our specifications are *modular*. That is, communication rules for channels and control flow rules for station programs are derived separately. Second, much of our analysis is *reusable*. Specifically, our proofs of timing relations for a particular bus network configuration are independent of station programs; they are not affected by changes in station programs. Furthermore, our proofs of protocol properties are independent of the number of stations and their positions on a bus.

Our reasoning is based upon the construction of a timed reachability graph using communication rules for channels and control flow rules for station programs. Protocol properties are proved by showing that they are invariant over time along every path in the TRG. While such a reasoning method is by no means simple—indeed, considerable human ingenuity is required—it is systematic and more reliable than what is available.

Acknowledgement: The authors thank the anonymous referees for their constructive comments. We are grateful to Ted Briggs for his invaluable suggestions and hours of fruitful discussions, to Thomas Woo for his suggestions on improvement to our notation, and to Chris Walton for his help with the figures.

References

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

[2] E. W. Dijkstra, "Guarded commands, non-determinacy, and formal derivation of programs," *Communications of the ACM*, vol. 18, pp 453-457, August 1975.

[3] 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.

[4] 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.

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

[6] M. Gouda, "Closed covers: To verify progress of communicating finite state machines," *IEEE Transactions on Software Engineering*, vol. SE-10, pp. 846-855, Nov. 1984.

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

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

[9] F. Jahanian and A. K. Mok, "Safety analysis of timing properties in real-time systems," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 9, pp.890-904, Sept. 1986.

[10] 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.

[11] P. Jain, "Specification of real-time broadcast networks," Ph. D. dissertation, Deptartment of Computer Sciences, University of Texas at Austin, 1991 (in preparation).

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

[13] S. S. Lam and A. U. Shankar, "A relational notation for state transition systems," *IEEE Transactions on Software Engineering*, vol. 16, no. 7, pp. 755-775, July 1990.

[14] G. M. Lundy and R. E. Miller, "A variable window protocol specification and analysis," *Proceedings 8th Int. Symp. on Protocol Specification, Testing, and Verification*, Atlantic City, June 1988.

[15] J. Misra and K. M. Chandy, "Proofs of networks of processes," *IEEE Transactions on Software Engineering*, vol. SE-7, pp. 417-426, July 1981.

[16] S. Owicki and L. Lamport, "Proving liveness properties of concurrent programs," *ACM Transactions on Programming Languages and Systems*, vol. 4, pp. 455-495, July 1982.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

Appendix A

Control Flow Rules of Modified Expressnet

Notation

$T_g = T_e + T_c + 2d + 2$

$\mathrm{EOC}\ (i, t) \equiv c_{out}(i, t-1) \wedge \neg c_{out}(i, t)$

$\mathrm{EOT}\ (i, t) \equiv c_{in}(i, t-d-2) \wedge \langle \forall t' : t-d-1 \le t' \le t : \neg c_{in}(i, t') \rangle$

CFR.1. $\mathbf{at}(i, S_1, u_1) \wedge \neg \text{request-to-wake-up}\ (i, u_1) \wedge \neg \text{talk}\ (i, u_1)$
$\Rightarrow$
$\mathbf{in}(i, S_1, u_1) \wedge \neg \text{talk}\ (i, u_1)$

CFR.2. $\mathbf{at}(i, S_1, u_1) \wedge \text{request-to-wake-up}\ (i, u_1) \wedge \neg \text{talk}\ (i, u_1)$
$\Rightarrow$
$\mathbf{after}(i, S_1, u_1) \wedge \neg \text{talk}\ (i, u_1)$

CFR.3. $\mathbf{in}(i, S_1, u_1) \wedge \neg \text{request-to-wake-up}\ (i, u_1+1) \wedge \neg \text{talk}\ (i, u_1)$
$\Rightarrow$
$\mathbf{in}(i, S_1, u_1+1) \wedge \neg \text{talk}\ (i, u_1+1)$

CFR.4. $\mathbf{in}(i, S_1, u_1) \wedge \text{request-to-wake-up}\ (i, u_1+1) \wedge \neg \text{talk}\ (i, u_1)$
$\Rightarrow$
$\mathbf{after}(i, S_1, u_1+1) \wedge \neg \text{talk}\ (i, u_1+1)$

The above four rules are derived from Axioms A12a-b. CFR.1 and CFR.2 have been refined by adding the same conjunct, $\neg$ talk (i, u_1), to both sides of each rule. Axiom A12c has been used to refine CFR.3 and CFR.4. From rule CFR.3 and using induction on time, we derive the following (aggregate) control flow rule.

CFR.3′. $\mathbf{in}(i, S_1, u_1) \wedge \langle \forall t' : u_1 \le t' \le u_2 : \neg \text{request-to-wake-up}\ (i, t') \rangle \wedge \neg \text{talk}\ (i, u_1)$
$\Rightarrow$
$\langle \forall t' : u_1 \le t' \le u_2 : \mathbf{in}(i, S_1, t') \wedge \neg \text{talk}\ (i, t') \rangle$

which will be used instead of CFR.3. Note that time parameter u_2 is free in CFR.3$'$. Some of the following control flow rules have been derived similarly using induction on time.

CFR.5. $\textbf{after}(i, S_1, u_1) \wedge \neg\, \text{talk}(i, u_1) \Rightarrow \textbf{at}(i, S_2, u_1) \wedge \neg\, \text{talk}(i, u_1)$

CFR.6. $\textbf{at}(i, S_2, u_1) \wedge \neg\, c_{in}(i, u_1) \wedge \neg\, \text{talk}(i, u_1) \Rightarrow \textbf{in}(i, S_2, u_1) \wedge \neg\, \text{talk}(i, u_1)$

CFR.7. $\textbf{in}(i, S_2, u_1) \wedge \neg\, \text{talk}(i, u_1) \wedge$
$\langle \exists t' : u_1 - T_g < t' \leq u_1 : \textbf{at}(i, S_2, t') \rangle \wedge \neg\, c_{in}(i, u_1 + 1)$
$\Rightarrow$
$\textbf{in}(i, S_2, u_1 + 1) \wedge \neg\, \text{talk}(i, u_1 + 1)$

CFR.8. $\langle \forall t' : u_1 \leq t' < u_1 + T_g : \textbf{in}(i, S_2, t') \rangle \wedge \neg\, c_{in}(j, u_1 + T_g) \wedge \neg\, \text{talk}(i, u_1)$
$\Rightarrow$
$\textbf{after}(i, S_2, u_1 + T_g) \wedge \text{asleep}(i, u_1 + T_g) \wedge \neg\, \text{talk}(i, u_1 + T_g)$

CFR.9. $\textbf{at}(i, S_2, u_1) \wedge c_{in}(i, u_1) \wedge \neg\, \text{talk}(i, u_1)$
$\Rightarrow$
$\textbf{after}(i, S_2, u_1) \wedge \text{awake}(i, u_1) \wedge \neg\, \text{talk}(i, u_1)$

CFR.10. $\textbf{in}(i, S_2, u_1) \wedge c_{in}(i, u_1 + 1) \wedge \neg\, \text{talk}(i, u_1)$
$\Rightarrow$
$\textbf{after}(i, S_2, u_1 + 1) \wedge \text{awake}(i, u_1 + 1) \wedge \neg\, \text{talk}(i, u_1 + 1)$

CFR.11. $\textbf{after}(i, S_2, u_1) \wedge \text{awake}(i, u_1) \wedge \neg\, \text{talk}(i, u_1) \Rightarrow \textbf{at}(i, S_4, u_1) \wedge \neg\, \text{talk}(i, u_1)$

CFR.12. $\textbf{after}(i, S_2, u_1) \wedge \text{asleep}(i, u_1) \Rightarrow \textbf{at}(i, S_3, u_1 + 1) \wedge \text{tx-pilot}(i, u_1 + 1)$

CFR.13. $\textbf{at}(i, S_3, u_1) \wedge \neg\, c_{in}(i, u_1) \wedge \text{tx-pilot}(i, u_1) \Rightarrow \textbf{in}(i, S_3, u_1) \wedge \text{tx-pilot}(i, u_1)$

CFR.14. $\textbf{in}(i, S_3, u_1) \wedge \langle \forall t' : u_1 < t' < u_2 : \neg\, c_{in}(i, t') \rangle \wedge \text{tx-pilot}(i, u_1)$
$\Rightarrow$
$\langle \forall t' : u_1 \leq t' \leq u_2 : \textbf{in}(i, S_3, t') \wedge \text{tx-pilot}(i, t') \rangle$

CFR.15. $\textbf{at}(i, S_3, u_1) \wedge c_{in}(i, u_1) \wedge \text{tx-pilot}(i, u_1) \Rightarrow \textbf{after}(i, S_3, u_1) \wedge \text{tx-pilot}(i, u_1)$

CFR.16. $\textbf{in}(i, S_3, u_1) \wedge c_{in}(i, u_1 + 1) \wedge \text{tx-pilot}(i, u_1)$
$\Rightarrow$
$\textbf{after}(i, S_3, u_1 + 1) \wedge \text{tx-pilot}(i, u_1 + 1)$

CFR.17. $\textbf{after}(i, S_3, u_1) \Rightarrow \textbf{at}(i, S_4, u_1 + 1) \wedge \neg\, \text{talk}(i, u_1 + 1)$

CFR.18. $\textbf{at}(i, S_4, u_1) \wedge \text{request-to-sleep}(i, u_1) \wedge \neg\, \text{talk}(i, u_1)$
$\Rightarrow$
$\textbf{at}(i, S_1, u_1) \wedge \neg\, \text{talk}(i, u_1)$

CFR.19. $\textbf{at}(i, S_4, u_1) \wedge \neg\, \text{request-to-sleep}(i, u_1) \wedge \neg\, \text{talk}(i, u_1)$
$\Rightarrow$
$\textbf{at}(i, S_5, u_1) \wedge \neg\, \text{talk}(i, u_1)$

CFR.20. $\textbf{at}(i, S_5, u_1) \wedge \neg\, \text{talk}(i, u_1) \wedge$
$\langle \forall t' : u_1 < t' \leq u_2 : \neg\, \text{EOC}(i, t') \rangle \wedge \langle \forall t' : u_1 + d + 1 < t' \leq u_2 : \neg\, \text{EOT}(i, t') \rangle$
$\Rightarrow$
$\langle \forall t' : u_1 \leq t' \leq u_2 : \textbf{in}(i, S_5, t') \wedge \neg\, \text{talk}(i, t') \rangle$

CFR.21. $\mathbf{in}(i, S_5, u_1) \ \wedge\ \neg\ \text{talk}\ (i, u_1) \ \wedge\ \langle \forall t' : u_1 < t' \leq u_2 : \neg\ \text{EOC}\ (i, t') \ \wedge\ \neg\ \text{EOT}\ (i, t') \rangle$

$\Rightarrow$

$\langle \forall t' : u_1 \leq t' \leq u_2 : \mathbf{in}(i, S_5, t') \ \wedge\ \neg\ \text{talk}\ (i, t') \rangle$

CFR.22. $\mathbf{in}(i, S_5, u_1) \ \wedge\ \neg\ \text{talk}\ (i, u_1) \ \wedge$

$\text{EOC}\ (i, u_1 + 1) \ \wedge\ \neg\ (\ \text{EOT}\ (i, u_1 + 1) \ \wedge\ \langle \forall t' : u_1 - d - 1 \ \leq t' \leq u_1 : \mathbf{in}(i, S_5, t') \rangle)$

$\Rightarrow$

$\mathbf{after}\ (i, S_5, u_1 + 1) \ \wedge\ \text{eoc}\ (i, u_1 + 1) \ \wedge\ \neg\ \text{talk}\ (i, u_1 + 1)$

CFR.23. $\langle \forall t' : u_1 \leq t' \leq u_1 + d + 1 : \mathbf{in}(i, S_5, t') \rangle \ \wedge\ \neg\ \text{talk}\ (i, u_1) \ \wedge\ \text{EOT}\ (i, u_1 + d + 2)$

$\Rightarrow$

$\mathbf{after}\ (i, S_5, u_1 + d + 2) \ \wedge\ \text{eot}\ (i, u_1 + d + 2) \ \wedge\ \neg\ \text{talk}\ (i, u_1 + d + 2)$

CFR.24. $\mathbf{after}\ (i, S_5, u_1) \ \wedge\ \text{eot}\ (i, u_1)$

$\Rightarrow$

$\langle \forall t' : u_1 < t' \leq u_1 + d : \text{tx-loco}\ (i, t') \rangle \ \wedge\ \mathbf{at}\ (i, S_6, u_1 + d)$

CFR.25. $\mathbf{after}\ (i, S_5, u_1) \ \wedge\ \text{eoc}\ (i, u_1) \ \wedge\ \neg\ \text{talk}\ (i, u_1) \ \Rightarrow\ \mathbf{at}\ (i, S_6, u_1) \ \wedge\ \neg\ \text{talk}\ (i, u_1)$

CFR.26*. $\mathbf{at}\ (i, S_6, u_1) \ \wedge\ \text{eot}\ (i, u_1) \ \wedge\ \neg\ \text{pts}\ (i, u_1)$

$\Rightarrow$

$\langle \forall t' : u_1 < t' \leq u_1 + d + 1 : \neg\ \text{talk}\ (i, t') \rangle \ \wedge\ \mathbf{at}\ (i, S_4, u_1 + d + 1)$

CFR.27. $\mathbf{at}\ (i, S_6, u_1) \ \wedge\ \text{eoc}\ (i, u_1) \ \wedge\ \neg\ \text{pts}\ (i, u_1) \ \wedge\ \neg\ \text{talk}\ (i, u_1)$

$\Rightarrow$

$\mathbf{at}\ (i, S_4, u_1) \ \wedge\ \neg\ \text{talk}\ (i, u_1)$

CFR.28. $\mathbf{at}\ (i, S_6, u_1) \ \wedge\ \text{pts}\ (i, u_1)$

$\Rightarrow$

$\langle \forall t' : u_1 < t' \leq u_1 + d + 1 : \text{tx-preamble}\ (i, t') \rangle \ \wedge\ \mathbf{at}\ (i, S_7, u_1 + d + 1)$

CFR.29. $\mathbf{at}\ (i, S_7, u_1) \ \wedge\ c_{\text{out}}\ (i, u_1) \ \Rightarrow\ \mathbf{at}\ (i, S_4, u_1 + 1) \ \wedge\ \neg\ \text{talk}\ (i, u_1 + 1)$

CFR.30. $\mathbf{at}\ (i, S_7, u_1) \ \wedge\ \neg\ c_{\text{out}}\ (i, u_1)$

$\Rightarrow$

$\langle \forall t' : u_1 < t' \leq u_1 + P_i : \text{tx-packet}\ (i, t') \rangle \ \wedge\ \mathbf{at}\ (i, S_4, u_1 + P_i + 1) \ \wedge$
$\neg\ \text{talk}\ (i, u_1 + P_i + 1)$

---

* For the original protocol, the corresponding control flow rule is:

OCFR.26. $\mathbf{at}\ (i, S_6, u_1) \ \wedge\ \text{eot}\ (i, u_1) \ \wedge\ \neg\ \text{pts}\ (i, u_1) \Rightarrow \mathbf{at}\ (i, S_4, u_1 + 1) \ \wedge\ \neg\ \text{talk}\ (i, u_1 + 1)$

Appendix B

Timed Reachability Graph of Modified Expressnet

Notation

t_0 : Free time parameter. Denotes the starting time of protocol execution.

t_1, t_2, t_3, t_4 : Time parameters, defined for notational convenience. (Each is defined as the sum of t_0 and a constant.)

$T_d = T_e + T_c + d$

$w_{ij} = t_i + \text{delay}_{1j}$

$v_{ij} = u_i + \text{delay}_{1j}$

P_j = packet transmission time for station j

inactive (j, t) $\equiv$ (**at** (j, S_1, t) $\vee$ **in** (j, S_1, t) $\vee$ **after** (j, S_1, t) $\vee$ **in** (j, S_2, t)) $\wedge$ $\neg$ talk (j, t)

idle (j, t) $\equiv$ (inactive (j, t) $\vee$ **at** (j, S_5, t) $\vee$ **in** (j, S_5, t)) $\wedge$ $\neg$ talk (j, t)

ready (j, t) $\equiv$ **at** (j, S_6, t) $\wedge$ pts (j, t)

quiescent (j, t) $\equiv \langle \forall t' : t - T_g \leq t' \leq t : \neg\, c_{in}(j, t') \rangle$

loco-sent (j, t) $\equiv \langle \forall t' : t - d < t' \leq t :$ tx-loco (j, t') $\rangle$

preamble-sent (j, t) $\equiv \langle \forall t' : t - d \leq t' \leq t :$ tx-preamble (j, t') $\rangle$

post-pilot (q, v_{1q}) $\equiv \langle \forall j :: \langle \forall t' : v_{1j} < t' \leq v_{1j} + d :$ idle (j, t') $\rangle \wedge c_{in}(j, v_{1j} + T_d) \wedge$
$\langle \forall t' : v_{1j} \leq t' \leq v_{1j} + T_d + d + 1 :$
$\neg$ EOT (j, t') $\wedge$ $\neg$ quiescent (j, t') $\rangle \rangle \wedge$
$\langle \forall j : 1 \leq j \leq q : \neg$ EOC (j, $v_{1j} + d + 1$) $\rangle \wedge$
$\langle \forall j : q < j \leq M :$ EOC (j, $v_{1j} + d + 1$) $\rangle$

Pre-RR (u_1) $\equiv \langle \forall j ::$ (**in** (j, S_1, v_{1j}) $\vee$ **at** (j, S_5, v_{1j}) $\vee$ **in** (j, S_5, v_{1j})) $\wedge$
$c_{in}(j, v_{1j}) \wedge \langle \forall t' : v_{1j} < t' \leq v_{1j} + T_d + d + 2 : \neg\, c_{in}(j, t') \rangle \wedge$
$\langle \forall t' : v_{1j} \leq t' \leq v_{1j} + d + 2 : \neg\, c_{out}(j, t') \rangle \rangle$

Init-RR (u_1) $\equiv \langle \exists k ::$ **after** (k, S_5, v_{1k}) $\wedge$ eot (k, v_{1k}) $\rangle \wedge$
$\langle \forall j ::$ (**after** (j, S_5, v_{1j}) $\wedge$ eot (j, v_{1j})) $\vee$
$\langle \forall t' : v_{1j} \leq t' \leq v_{1j} + 2d + 1 :$ idle (j, t') $\rangle \rangle \wedge$
$\langle \forall j, t' : v_{1j} < t' \leq v_{1j} + T_d : \neg$ EOT (j, t') $\wedge$ $\neg$ quiescent (j, t') $\rangle$

clear-packet (r, u_1) $\equiv$
$\langle \forall j : 1 \leq j < r : \neg$ ready (j, v_{1j}) $\rangle \wedge$ ready (r, v_{1r}) $\wedge$
$\langle \forall j ::$ (ready (j, v_{1j}) $\wedge$ preamble-sent (j, $v_{1j} + d + 1$) $\wedge$ **at** (j, S_7, $w_{1j} + d + 1$)) $\vee$
($\neg$ ready (j, v_{1j}) $\wedge$
$\langle \forall t' : v_{1j} \leq t' \leq v_{1j} + d + 1 : \neg$ talk (j, t') $\rangle \wedge$ idle (j, $v_{1j} + d + 1$)) $\rangle \wedge$
$\langle \forall j, t' : v_{1j} \leq t' \leq v_{1j} + T_d : \neg$ EOT (j, t') $\wedge$ $\neg$ quiescent (j, t') $\rangle$

$$\begin{aligned} B1\,(j, v_{1j}) \equiv\ & \text{preamble-sent}\,(j, v_{1j} + d + 1) \ \wedge \\ & \langle \forall t' : v_{1j} + d + 2 \ \leq t' \leq v_{1j} + P_j + d + 1 : \text{tx-packet}\,(j, t') \rangle \ \wedge \\ & \text{idle}\,(j, v_{1j} + P_j + d + 2) \end{aligned}$$

$$\begin{aligned} B2\,(j, v_{1j}) \equiv\ & \langle \forall k : 1 \ \leq k \ < j : \\ & \qquad \langle \forall t' : v_{1k} < t' \leq v_{1k} + d + 1 : \neg\ \text{talk}\,(k, t') \rangle \rangle \ \wedge \\ & \langle \forall k : j \neq k : \\ & \qquad \langle \forall t' : v_{1k} + d + 2 \ \leq t' \leq v_{1k} + P_j + 2d + 1 : \text{idle}\,(k, t') \rangle \rangle \end{aligned}$$

$$\begin{aligned} B3\,(j, v_{1j}) \equiv\ & \langle \forall k, t' : v_{1k} < t' \leq v_{1k} + P_j + T_d + 2d + 2 : \\ & \qquad \neg\ \text{EOT}\,(k, t') \ \wedge \ \neg\ \text{quiescent}\,(k, t') \rangle \end{aligned}$$

Indices q and m have the following meaning:

q : Left-most station to transmit a pilot (same in formulas F9 through F11).

m : Station that has finished transmitting its packet. Used in the induction step
(same in formulas F27 through F34).

Note that m and q are free variables of TRG. Therefore, properties proved using the TRG are valid for all possible values of m and q. The time parameter t_0 occurs free in all formulas F1 through F25. Formulas F26 through F33 correspond to the induction step, and time parameter t_5 occurs free in them.

Formulas describing (aggregate) nodes of the TRG

$$\begin{aligned} \mathbf{F1} \equiv\ & \langle \forall j :: \mathbf{in}\,(j, S_1, t_0 - 1) \ \wedge \ \neg\ \text{talk}\,(j, t_0 - 1) \ \wedge \\ & \qquad \langle \forall t' : t_0 - 1 \ \leq t' < t_0 + T_d : \neg\ c_{in}\,(j, t') \rangle \rangle \end{aligned}$$

$$\mathbf{F2} \equiv \mathbf{F1} \ \wedge \ \langle \forall j :: \neg\ \text{request-to-wake-up}\,(j, t_0) \rangle$$

$$\mathbf{F3} \equiv \mathbf{F1} \ \wedge \ \langle \exists j :: \text{request-to-wake-up}\,(j, t_0) \rangle$$

$$\mathbf{F4} \equiv \langle \forall j :: \mathbf{in}\,(j, S_1, t_0) \ \wedge \ \neg\ \text{talk}\,(j, t_0) \ \wedge \ \langle \forall t' : t_0 \leq t' \leq t_0 + T_d : \neg\ c_{in}\,(j, t') \rangle \rangle$$

$$\begin{aligned} \mathbf{F5} \equiv\ & \langle \forall j :: (\ \mathbf{in}\,(j, S_1, t_0) \ \vee \ \mathbf{after}\,(j, S_1, t_0)\ ) \ \wedge \ \neg\ \text{talk}\,(j, t_0) \ \wedge \\ & \qquad \langle \forall t' : t_0 \leq t' \leq t_0 + T_d : \neg\ c_{in}\,(j, t') \rangle \rangle \ \wedge \\ & \langle \exists j :: \mathbf{after}\,(j, S_1, t_0) \rangle \end{aligned}$$

$$\begin{aligned} \mathbf{F6} \equiv\ & \langle \forall j :: \langle \forall t' : t_0 \leq t' < t_0 + T_g : \text{inactive}\,(j, t') \rangle \ \wedge \\ & \qquad \langle \forall t' : t_0 \leq t' < t_0 + T_g : \neg\ c_{in}\,(j, t') \rangle \rangle \ \wedge \\ & \langle \exists j :: \langle \forall t' : t_0 \leq t' < t_0 + T_g : \mathbf{in}\,(j, S_2, t') \rangle \rangle \end{aligned}$$

$$\begin{aligned} \mathbf{F7} \equiv\ & \langle \exists k, t_1 : t_0 + T_g - \text{delay}_{1k} \leq t_1 \leq t_0 + T_g : \\ & \qquad (\ \mathbf{after}\,(k, S_2, w_{1k}) \ \wedge \ \text{asleep}\,(k, w_{1k}) \ \wedge \ \neg\ \text{talk}\,(k, w_{1k})\ ) \ \wedge \\ & \qquad \langle \forall j :: \langle \forall t' : t_0 + T_g \leq t' < w_{1j} : \text{inactive}\,(j, t') \rangle \ \wedge \\ & \qquad\qquad \langle \forall t' : w_{1j} \leq t' \leq w_{1j} + T_d : \neg\ c_{in}\,(j, t') \rangle \rangle \rangle \end{aligned}$$

In the following formulas, t_1 is existentially quantified over the range:
$t_0 + T_g - T_e \leq t_1 \leq t_0 + T_g$.

$$\mathbf{F8} \equiv \langle \forall j :: \langle \exists u_1 : w_{1j} \le u_1 \le w_{1j} + T_d : \langle \forall t' : w_{1j} \le t' \le u_1 : \text{inactive}(j, t') \lor (\textbf{after}(j, S_2, t') \land \text{asleep}(j, t') \land \neg\, \text{talk}(j, t'))\rangle \land \langle \forall t' : u_1 < t' \le w_{1j} + T_d : \textbf{in}(j, S_3, t') \land \text{tx-pilot}(j, t')\rangle\rangle\rangle \land \langle \exists k :: \langle \forall t' : w_{1k} < t' \le w_{1k} + T_d : \textbf{in}(k, S_3, t') \land \text{tx-pilot}(k, t')\rangle\rangle$$

$$\mathbf{F9} \equiv \langle \exists q :: \text{post-pilot}(q, w_{1q} + T_d + 1)\rangle$$

In the following formulas, $t_2 = t_1 + T_d + 1$

$$\mathbf{F10} \equiv \mathbf{F9} \land \langle \forall j : q < j \le M : \neg\, \text{ready}(j, w_{2j} + d + 1)\rangle$$
$$\mathbf{F11} \equiv \mathbf{F9} \land \langle \exists j : q < j \le M : \text{ready}(j, w_{2j} + d + 1)\rangle$$
$$\mathbf{F12} \equiv \text{Pre-RR}(t_2 + T_d) \land \langle \forall j, t' : w_{2j} < t' \le w_{2j} + T_d : \neg\, \text{talk}(j, t')\rangle$$

In the following formulas, $t_3 = t_2 + T_d$.

$$\mathbf{F13} \equiv \mathbf{F12} \land \langle \forall j :: \textbf{in}(j, S_1, w_{3j})\rangle$$
$$\mathbf{F14} \equiv \mathbf{F12} \land \langle \exists j :: \textbf{at}(j, S_5, w_{3j}) \lor \textbf{in}(j, S_5, w_{3j})\rangle$$
$$\mathbf{F15} \equiv \mathbf{F13} \land \langle \exists k, u_1 : t_3 \le u_1 \le t_3 + \text{delay}_{kM} : \textbf{after}(k, S_1, v_{1k}) \land \neg\, \text{talk}(k, v_{1k}) \land \langle \forall j, t' : w_{3j} \le t' < v_{1j} : \textbf{in}(j, S_1, t') \land \neg\, \text{talk}(j, t')\rangle\rangle$$
$$\mathbf{F16} \equiv \mathbf{F13} \land \langle \forall j, t' : w_{3j} \le t' \le t_3 + \text{delay}_{1M} : \textbf{in}(j, S_1, t') \land \neg\, \text{talk}(j, t')\rangle$$
$$\mathbf{F17} \equiv \langle \exists k, u_2 : t_3 + T_g \le u_2 \le t_3 + T_g + \text{delay}_{kM} : (\textbf{after}(k, S_2, v_{2k}) \land \text{asleep}(k, v_{2k}) \land \neg\, \text{talk}(k, v_{2k})) \land \langle \forall j :: \text{inactive}(j, v_{2j} - 1) \land \langle \forall t' : v_{2j} \le t' \le v_{2j} + T_d : \neg\, c_{in}(j, t')\rangle\rangle\rangle$$
$$\mathbf{F18} \equiv \langle \forall j :: \textbf{in}(j, S_1, t_3 + \text{delay}_{1M}) \land \langle \forall t' : t_3 + \text{delay}_{1M} \le t' \le t_3 + \text{delay}_{1M} + T_d : \neg\, c_{in}(j, t')\rangle \land \langle \forall t' : w_{3j} \le t' \le t_3 + \text{delay}_{1M} : \neg\, \text{talk}(j, t')\rangle\rangle$$
$$\mathbf{F19} \equiv \text{Init-RR}(t_3 + d + 2) \land \langle \forall j, t' : w_{3j} \le t' \le w_{3j} + d + 2 : \neg\, \text{talk}(j, t')\rangle$$

In the following formulas, $t_4 = t_3 + d + 2$.

$$\mathbf{F20} \equiv \langle \exists k :: \text{loco-sent}(k, w_{4k} + d)\rangle \land \langle \forall j :: ((\text{loco-sent}(j, w_{4j} + d) \land \textbf{at}(j, S_6, w_{4j} + d) \land \text{eot}(j, w_{4j} + d)) \lor \langle \forall t' : w_{4j} \le t' \le w_{4j} + 2d + 1 : \text{idle}(j, t')\rangle) \land c_{in}(j, w_{4j} + T_d + d) \land \langle \forall t' : w_{4j} < t' \le w_{4j} + T_d + d : \neg\, \text{EOT}(j, t') \land \neg\, \text{quiescent}(j, t')\rangle\rangle$$
$$\mathbf{F21} \equiv \mathbf{F20} \land \langle \forall j :: \neg\, \text{ready}(j, w_{4j} + d)\rangle$$
$$\mathbf{F22} \equiv \mathbf{F20} \land \langle \exists j :: \text{ready}(j, w_{4j} + d)\rangle$$
$$\mathbf{F23} \equiv \text{Pre-RR}(t_4 + T_d + d) \land \langle \forall j, t' : w_{4j} + d < t' \le w_{4j} + T_d + d : \neg\, \text{talk}(j, t')\rangle$$

$$\mathbf{F24} \equiv \langle \exists k :: \text{clear-packet}(k, w_{4k} + d)\rangle$$

F25 $\equiv$ $\langle\, \exists k :: \text{clear-packet}\ (k,\ w_{2k} + d + 1)\,\rangle$

The following formulas correspond to the induction step.

F26 $\equiv$ $\text{clear-packet}\ (m,\ t_5 + \text{delay}_{1m})$

F27 $\equiv$ $\text{clear-packet}\ (m,\ t_5 + \text{delay}_{1m})\ \wedge\ B1\ (m,\ w_{5m})$

F28 $\equiv$ $B1\ (m,\ w_{5m})\ \wedge\ B2\ (m,\ w_{5m})\ \wedge\ B3\ (m,\ w_{5m})$

F29 $\equiv$ $\text{post-pilot}\ (m,\ w_{5m} + P_m + d + 1)$

In the following formulas, $t_6 = t_5 + P_m + d + 1$.

F30 $\equiv$ **F29** $\wedge\ \langle\, \forall j : m < j \leq M : \neg\ \text{ready}\ (j,\ w_{6j} + d + 1)\,\rangle$

F31 $\equiv$ **F29** $\wedge\ \langle\, \exists j : m < j \leq M : \text{ready}\ (j,\ w_{6j} + d + 1)\,\rangle$

F32 $\equiv$ $\text{Pre-RR}\ (t_6 + T_d)\ \wedge\ \langle\, \forall j, t' : w_{6j} < t' \leq w_{6j} + T_d : \neg\ \text{talk}\ (j,\ t')\,\rangle$

F33 $\equiv$ $\langle\, \exists k :: \text{clear-packet}\ (k,\ w_{6k} + d + 1)\,\rangle$

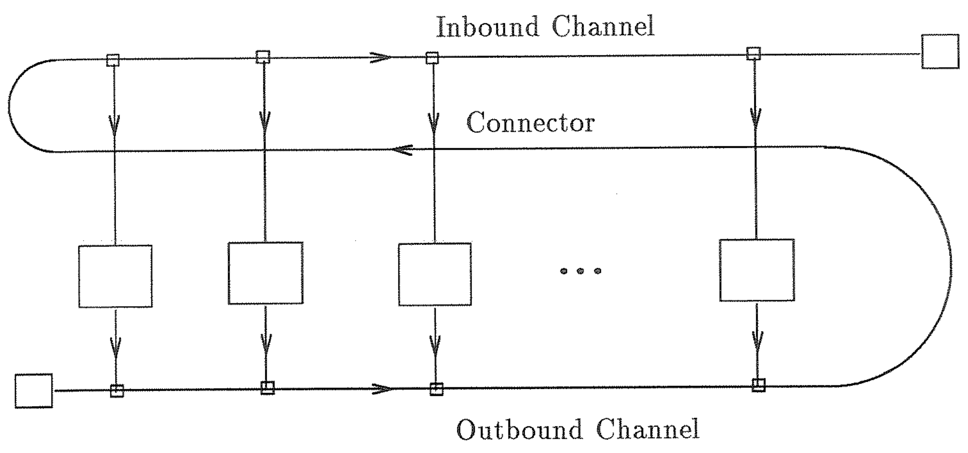

Figure 1: UBS with a single folded cable

```
while true do
  begin
    {$S_1$} wait (request-to-wake-up);
    {$S_2$} wait-par                                        (* Check if net is awake *)
                wait ($c_{in}$) :: awake
              || wait-seq (not $c_{in}$ for $T_e + T_c + 2d + 3$) :: asleep
          end-wait-par;
          if asleep then                   (* Cold Start *)
             begin
                   set tx-pilot, talk := true, true;
               {$S_3$} wait ($c_{in}$);
                   set tx-pilot, talk := false, false;
             end;
    {$S_4$} while not request-to-sleep do
             begin
               {$S_5$} wait-par
                         wait-seq ($c_{in}$ ; not $c_{in}$ for $d + 2$) :: eot
                       || wait-seq ($c_{out}$ ; not $c_{out}$) :: eoc
                   end-wait-par;
                   if eot then
                      begin                (* Start a new train *)
                        set tx-loco, talk := true, true;
                        delay ($d - 1$)
                      end;
              {$S_6$} if not packet-to-send then
                        begin
                          if eot then
                             set tx-loco, talk := false, false;
                        end
                    else
                       begin              (* Attempt to transmit packet *)
                         set tx-loco, tx-preamble, talk := false, true, true;
                         delay (d);
                      {$S_7$} if $c_{out}$ then            (* Defer to upstream station *)
                                set tx-preamble, talk := false, false
                             else
                                begin             (* Complete transmission of TU *)
                                  set tx-preamble, tx-packet, talk := false, true, true;
                                  delay (tx-delay $- 1$);
                                  set tx-packet, talk := false, false
                                end
                       end
             end
  end
```

Figure 2: Station program for Expressnet

```
{$S_4$} while not request-to-sleep do
        begin
            {$S_5$} wait-par
                        wait-seq ($c_{in}$ ; not $c_{in}$ for d + 2) :: eot
                      || wait-seq ($c_{out}$ ; not $c_{out}$ ) :: eoc
                    end-wait-par;
                    if eot then
                        begin                   (* Start a new train *)
                            set tx-loco, talk := true, true;
                            delay (d - 1)
                        end;
           {$S_6$} if not packet-to-send then
                            begin
                                if eot then
                                    begin
                                        set tx-loco, talk := false, false;
                                        delay (d)          (* MODIFICATION for collision-freedom *)
                                    end
                            end
                     else
                         begin              (* Attempt to transmit packet *)
                             set tx-loco, tx-preamble, talk := false, true, true;
                             delay (d);
                       {$S_7$} if $c_{out}$ then              (* Defer to upstream station *)
                                     set tx-preamble, talk := false, false
                                 else
                                     begin              (* Complete transmission of TU *)
                                         set tx-preamble, tx-packet, talk := false, true, true;
                                         delay (tx-delay - 1);
                                         set tx-packet, talk := false, false
                                     end
                         end
        end
```

Figure 3: Modified station program for Expressnet (regular round only)

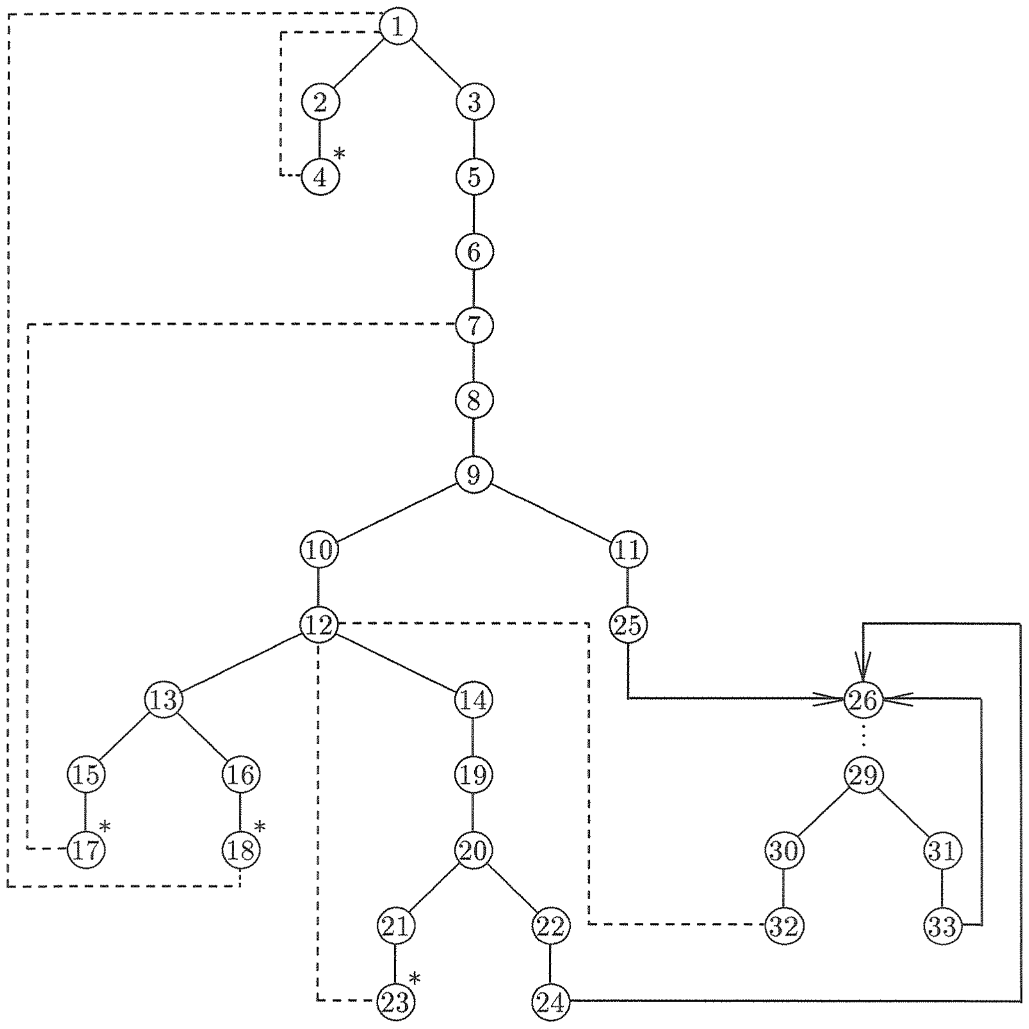

Figure 4: Timed reachability graph of modified Expressnet