A Timing-based Schema for Stabilizing Information Exchange

Anish Arora * *David M. Poduska*

Abstract

The paradigm of information exchange provides a basis for nodes in a network to stay uptodate with the recent information in the network. In this paradigm, nodes cooperate with each other to share their current information. We present a simple and uniform schema for building information exchange protocols that are stabilizing, in the following strong sense. Starting from arbitrary state, the protocols reach within bounded real-time a state wherefrom all nodes remain uptodate with the recent information in the network.

The ability to stabilize in bounded time is achieved by using timing-based actions. The timing constraints on these actions can be systematically adapted to suit a variety of network loads, delay requirements, and scheduling restrictions and to tolerate out-of-phase and drift-prone node clocks. Our schema also tolerates any number of topological changes in the network. Moreover, it accommodates information that is time-varying as well as it does information that is fixed. It is thus well-suited to dynamic high speed networks.

Keywords: stabilization, timing-based protocols, fault-tolerance, adaptivity, high speed networks, formal methods, verification

1 Introduction

It is often necessary for nodes in a computer network to exchange information with each other. Not only do such "information exchange" protocols enable nodes to collect, disseminate, or update information, they also provide a basis to solve diverse problems in distributed computation, communication, and control.

If networks were stable and all nodes could execute simultaneously, solutions to information exchange problems would be designed readily: in a sequence of simultaneous steps, network nodes would propagate the information they noted in the last step and make note of the information received in current step.

Most networks are, however, dynamic. As a result, solutions to information exchange problems become more complex. Consider, for example, a network where nodes and channels may added or be removed at any time. When a node is removed, information that could be communicated solely via that node has to be removed from the system in bounded time. Likewise, when a channel is removed, information that could be communicated solely via that channel has to be removed from the system in bounded time. In addition, when a node or channel is added, the duals of the abovementioned considerations have to be dealt with.

*Dept. Comp. Sc., The Ohio State University, Columbus, OH 43210, {anish, poduska}@cis.ohio-state.edu. Supported in part by NSF Grant CCR-93-08640 and OSU Grant 221506

Moreover, network nodes can not execute simultaneously. As a result, even when nodes execute at exactly the same speed, an out-of-phase node may execute incorrectly, for example, by removing some information just before renewed confirmation of that information arrives. The situation is further complicated by the variable communication delay of channels and by the drift in the speeds of nodes (both from the ideal clock and from each other).

Finally, networks today operate at high speeds and aim to provide small end-to-end delay to applications. As a result, the use of protocols that explicitly acknowledge receipt of information is undesirable; and, loss of coordination is better addressed within the scope of protocols than as a separate task. Modern networks are also increasingly subject to wide variation in network load, quality of service requirements, and restrictions on node schedulers. As a result, protocols that can be adapted to suit a variety of network situations are desirable.

Contribution of the paper. In this paper, we focus our attention on the entire class of information exchange problems, including ones in which the information varies over time. We systematically develop a simple and uniform protocol schema for designing information exchange protocols that are stabilizing, in the following strong sense. Starting from arbitrary state, the protocols reach within bounded real-time a state wherefrom all nodes remain uptodate with the recent information in the network.

The ability to stabilize in bounded time is achieved by using timing-based actions. The timing constraints on these actions can be systematically adapted to suit a variety of network loads, delay requirements and scheduling restrictions, and to tolerate out-of-phase and drift-prone node clocks. Our protocol schema is also able to tolerate any number of topological changes in the network, by ensuring that when the topology stabilizes the protocols stabilize to a state wherefrom all up nodes remain uptodate with the recent information in the up portion of the network.

We also describe in this paper a simple and uniform verification schema to demonstrate the correctness of the resulting protocols.

Previous Work. Gouda and Multari [1] present an indepth study of stabilization in networks protocols, although the protocols they consider are not explicitly timing-based. In fact, we have found only a few stabilizing protocols in the literature that are explicitly timing-based. Varghese [2] mentions that propagation of information with feedback may be achieved in a stabilizing fashion using timing-based actions on a tree network. He also refers to timing-based stabilizing solutions for maintaining spanning trees (due to Perlman), data links and virtual circuits (due to Spinelli).

As regards verification methods, our approach builds on previous work [3-12]. We restrict our proofs of timing properties to the use of two bounded temporal concepts, namely bounded response and bounded invariance. We use timing properties essentially to exhibit bounds on the convergence time and to deduce untimed properties.

A discussion of information exchange protocols appears in Segall [13]. His protocols are not stabilizing. They do, however, tolerate topological changes, although unlike our protocols, they use messages of unbounded size to achieve fault-tolerance.

Overview of the paper. To motivate our schema, we begin by presenting a stabilizing timing-based solution to the adjacency problem, a problem that is basic to distributed computer networks. In this problem, network nodes need to know the nodes that are "adjacent" to them. A node is adjacent to another node iff both nodes are "up" and there is an up channel between them.

We then show that our solution to the adjacency problem can be elegantly generalized to solve a more general information exchange problem, namely the connectivity problem. In this problem, network nodes need to know the nodes that are "connected" to them. A node is "connected" to another node iff there is a path from the former to the latter that consists solely of up nodes and up channels. Thus, whereas the adjacency problem only requires information about nodes to be communicated to their neighbors, the connectivity problem requires that information about nodes be communicated to all nodes in the system that are reachable from them.

We next show that the structure of the connectivity problem can be generalized to solve all information exchange problems. The factors that may be considered in this generalization include:

- *Type*: is the information untyped, Boolean, Ids, Integer, Real, Space Coordinate, Time Coordinate, or a composite type,
- *Domain*: is the information exchange within a neighborhood, between a subset of nodes, or in the entire network,
- *Fixed/Variable*: how does the information vary over time,
- *Frequency*: how often does the information need to be exchanged,
- *Consistency*: how do nodes determine whether the information received in the exchange is current,
- *Authentication*: how do nodes determine whether the information received in the exchange is authentic,
- *Application*: how do nodes use the information received in the exchange.

The resulting program schema can be instantiated to solve diverse information exchange problems. With untyped information, for example, the problems of Adjacency and Connectivity are solved. With Boolean-valued information, the problems of Consensus and Commitment are solved. With Id-valued information, Leader Election and Spanning Tree Construction problems are solved. With Integer or Real-valued information, Maxima Finding and Routing problems are solved. With Space Coordinate information, Mobility and Geometry problems can be solved. And, with Time Coordinate information, Distributed Simulation and Clock Synchronization problems can be solved.

Finally, we discuss how our schema can be adapted to suit a variety of network loads, delay requirements, and scheduling restrictions.

Organization of the paper. In Section 2, we discuss notation for timing-based protocols and for temporal modalities used to verify timing-based protocols. In Section 3, we present a stabilizing solution to the adjacency problem and verify its correctness. In Section 4, we generalize the solution and verification given in Section 3 to solve the connectivity problem. In Section 5, we generalize the solution and verification given in Section 4 to a schema for solving information exchange problems. In Section 6, we discuss the adaptivity of our protocol schema. Finally, conclusions follow in Section 7.

2 Timing-Based Protocols

2.1 Syntax and Semantics

A timing-based protocol consists of a set of variables, a set of actions, and a set of timing constraints. Each variable has a predefined nonempty domain. Each action has the form

$$guard \xrightarrow{[L,U)} statement$$

where the guard is a boolean expression over protocol variables, L and U are constant timebounds such that $0 \leq L$ and $L < U$, and the statement is a terminating update of protocol variables. Each timing constraint is an arithmetic relation between the timebounds of the protocol actions.

Let p be a timing-based protocol. A *state* of p is defined by a value for each variable of p, chosen from the domain of the variable. An action of p is *enabled* at a state iff its guard is true at that state. A *state predicate* of p is defined by a boolean expression over the variables of p.

Based on the semantics of maximal parallelism, a *computation* of p schedules execution of the statement of each action as soon as the guard of that action is enabled. The time elapsed from the guard of an action being enabled to the statement of the action being executed is within the interval $[L, U)$ of the action. Each statement is executed in an atomic step. Note that the guard of the action may no longer be enabled when the statement is executed.

More formally, a computation of p is a fair sequence of steps: in each step, the statement of some action j is executed. The statement is chosen such that the following two conditions hold: (i) since the last step in which j was executed, j has been enabled at least once, and (ii) the time between the earliest such state where j was enabled and the current state is in the range $[L, U)$. By fairness of the computation, we mean that if an action is enabled at a state, the statement of the action is subsequently executed in the sequence.

A state predicate S is *closed* in p, informally, iff in every computation of p, once S holds it continues to hold. Recall that in untimed protocols the verification of closure is relatively easy: Since guard evaluation and statement execution occur in the same instant, closure is verified by showing for each action of p that at any state where the action is enabled and S holds, executing the statement of the action yields a state where S holds. By way of contrast, in timed protocols the verification of closure is more complicated: Since there is a delay between guard evaluation and statement execution, closure is verified by showing for each action of p that at any state in a computation where S holds, either executing the action yields a state where S holds or the action cannot have been enabled at any past time within the upper timebound of the action. We meet this obligation by exhibiting for each action a predicate P such that in any state where $S \wedge P$ holds, executing the statement preserves S; and in any state where $S \wedge \neg P$ holds, the statement was not enabled at any past time within the upper timebound of the action.

More formally, S is *closed* in p iff for each action $g \xrightarrow{[L,U)} st$, $(\exists P :$
$(S \wedge \neg P) \;\Rightarrow\; (\neg g$ has held for at least past U time$)$:
$\{S \wedge P\}\; st \;\{S\})$

Note that when time is ignored the above definition of closure reduces to the familiar untimed definition: for each action $g \xrightarrow{[L,U)} st$,
$\{S \wedge g\}\; st \;\{S\}$.

A timing-based protocol p is *stabilizing* for a closed state predicate S iff upon starting from an arbitrary state every computation of p reaches within bounded time a state where S holds.

2.2 Verification

An established method for verifying the correctness of untimed protocols is to exhibit a state invariant. Intuitively, the state invariant of a protocol is a state predicate that characterizes the "intended" states of protocol execution [4, 14, 6]. Thus, every computation of an untimed protocol that starts at a state where its state invariant holds is an intended computation, one that meets the (safety and progress properties of the) problem specification that the protocol satisfies.

This method of state invariants remains valid for timing-based protocols. However, the task of exhibiting the state invariant of timing-based protocols requires knowledge of timing dependencies in the protocol. These timing dependencies can be incorporated within the state invariant, as in [7, 8], or within a separate assertion containing timing properties [9]. We prefer to separate timing assertions from state assertions: first, we exhibit a timing invariant, named $\mathcal{TI}$, and then using $\mathcal{TI}$ we exhibit a state invariant, named $\mathcal{SI}$.

The timing invariant $\mathcal{TI}$ of a protocol asserts that certain timing assertions are true in every computation of the protocol, regardless of the states that the computations start in. $\mathcal{TI}$ is used only to verify that $\mathcal{SI}$ is closed; i.e, $\mathcal{TI}$ is used to show that statements that may falsify $\mathcal{SI}$ are restricted to execute only in those states where they preserve $\mathcal{SI}$. Once SI is validated, safety and progress progress properties of the protocol are verified just as they are in untimed programs.

The correctness of $\mathcal{TI}$ relies on the time bounds of the actions and the timing constraints placed on those bounds. Given the time bounds and the timing constraints, $\mathcal{TI}$ is proven nonoperationally by examining each protocol action.

In our notation, $\mathcal{TI}$ consists of two types of timing assertions, one involving "bounded invariance" — some property has remained stable for a period of time — and the other involving "bounded response" — from a given state something is guaranteed to occur within a bounded amount of time.

We use two temporal modalities to permit assertions of these two types of timing assertions. To specify a bounded invariance property such as, "if a siren sounds an error condition has persisted for 10 seconds", we write

$$siren \;\Rightarrow\; (\lhd\!\Box : 10secs : error).$$

To specify a bounded response property such as "if an error is detected then an alarm will sound within 10 seconds", we write

$$error \;\Rightarrow\; (\rhd : 10secs : alarm).$$

Note that as given the intervals over which the two operators are defined are not duals; the $\rhd$ operator provides an open, future interval (e.g., in less than 10 seconds) while the $\lhd\!\Box$ operator provides a closed, past interval (e.g., for at least 10 seconds). We choose these two operators because they are well understood and suffice for our purposes. Note also that the $\rhd$ operator is equivalent to a bounded version of the $\mathcal{F}$ operator and the $\lhd\!\Box$ operator corresponds to a bounded version of the $\mathcal{S}$ operator in temporal logic [11, 15].

2.3 Network Assumptions

A computer network consists of N nodes and some number of communication channels that each connect a unique pair of nodes. Channels are bidirectional: the channel directed from an arbitrary node j to an arbitrary node k is denoted $\langle j,k\rangle$ and the channel directed from node k to node j is denoted $\langle k,j\rangle$. At each instant, a predicate $up.j$ is true iff node j is up, and a predicate $up.\langle j,k\rangle$ is true iff $\langle j,k\rangle$ is up. Actions of node j may communicate with node k iff $up.j$ and $up.\langle j,k\rangle$ hold. The communication time along any directed channel is guaranteed to be less than R, where $R > 0$.

For convenience, we use a standard action to represent the communication on each directed channel. If channel $\langle k,j\rangle$ is non-empty, denoted $ch.k.j \neq \langle\rangle$ and the recipient node j is up, then within R time units of information being sent, the information is received into buffer $q.j.k$ at node j and the channel is cleared. Formally, the "channel" action for $\langle k,j\rangle$ is:

$$ch.k.j \neq \langle\rangle \wedge up.j \;\xrightarrow{[0,R)}\; q.j.k, ch.k.j := last.(ch.\langle k,j\rangle), \langle\rangle$$

Note that some information may be "lost" if, for instance, information is received twice during an iteration cycle.

Up nodes and channels may suffer fail-stop failure at any time. Down nodes and channels may repair with an arbitrary state at any time. When a channel repairs, both of its directed channels repair, although not necessarily at the same time.

3 A Stabilizing Timing-Based Adjacency Protocol

Specification: It is required to design for each node j a set of actions that maintain for each channel $\langle j,k\rangle$ a variable $adj.j.k$, for k is adjacent to j, such that $adj.j.k$ is true iff j can prove that $up.k$ and $up.\langle j,k\rangle$ hold.

Design: A standard approach to the adjacency problem is for each node to periodically send "keep-alive" messages to its neighbors to inform them that it is still up; if a neighbor does not receive a keep-alive message from the node within some number of periods, the neighbor can assume the node has failed.

We design each node to first send a keep-alive message to each neighbor and then determine the age of the last keep-alive message received from each neighbor. When the age of the last keep-alive message equals a threshold K, that neighbor is marked as being nonadjacent. This is achieved by an action that executes as follows. First, node j appends some information to each outgoing channel $ch.j.k$ to inform each neighbor k that j is up. Exactly what information is appended is of no consequence, so we append arbitrary information (denoted "?"). Second, each incoming $q.j.k$ is polled to determine whether a keep-alive message was received in the last iteration (note that since the actual information that is received is of no consequence, we refer to $q.j.k$ as a boolean instead of using $q.j.k \neq \langle\rangle$ and $q.j.k = \langle\rangle$). Finally, the local adjacency relation is updated in one of three ways:

1. If node j received a keep-alive message from node k in the previous round, we indicate node j has current information that node k is adjacent to node j by resetting variable $age.j.k$ to 0 and truthifying $adj.j.k$. The domain of $age.j.k$ is $\{0 \ldots K\}$.
2. If node j did not receive a keep-alive message from node k in the previous round, we "age" j's local knowledge of k's adjacency; that is, we say node j's information regarding node k is less current than it was the previous round.
3. If node j has not received any keep-alive messages from node k in any of the previous K rounds, we assume that either node k or channel $ch.k.j$ has failed. That information asserted by node k, namely the fact that k is adjacent, is subsequently retracted by falsifying $adj.j.k$.

Formally, this "node" action for node j is:

$$
\begin{array}{l}
up.j \xrightarrow{[S,T)} \\
(\| \; k : up.\langle k,j\rangle : \\
\quad ch.j.k := ch.j.k \;\cdot\; ? \\
\;;\; \mathbf{if} \\
\quad\quad q.j.k \;\rightarrow\; adj.j.k, age.j.k, q.j.k := true, 0, false \\
\quad [] \\
\quad\quad \neg q.j.k \wedge age.j.k < K \;\rightarrow\; age.j.k := age.j.k + 1 \\
\quad [] \\
\quad\quad \neg q.j.k \wedge age.j.k = K \;\rightarrow\; adj.j.k := false \\
\quad \mathbf{fi})
\end{array}
$$

The time bounds on the node action capture both the bound on the rate of the fastest processor to the slowest processor and the varying speeds of nodes: A fast node may only require S time to complete an iteration while a slower node may require up to T time to complete an iteration. The speed of a node may change over time, completing one iteration in as little as S time and another iteration in up to T time.

Our design ensures that if each node completes its iteration within the interval $[S,T)$, then provided the timing constraint $\mathbf{R + T < KS}$ is satisfied, every computation of the adjacency protocol upon starting from an arbitrary state converges to a state where the adjacency relation is correct and remains correct.

3.1 Verification

Intuitively, the correctness of each node's information relies on a node receiving a keep-alive message from each up neighbor before age reaches K. The concern is that a fast node will falsely mark a slower neighbor down before the neighbor has a chance to send the node a message. By examining the protocol, we know that (i) a node j can only mark a neighbor k down after $age.j.k = K$ and (ii) an up node broadcasts its status at most T time after its previous broadcast, and the information is received at most R time after being sent. It hence follows from the timing constraint $R + T < KS$ that no node is marked down falsely. Again, this constraint is part of the correctness criteria — in order for an implementation to be correct, the implementation must satisfy the timing constraint.

Timing Invariant:

$$
\begin{array}{lll}
\mathcal{TI} & \equiv & (\forall j,k : up.j : \\
& & \quad (up.k \wedge up.\langle k,j\rangle) \;\Rightarrow\; (\rhd : KS : q.j.k) \\
& & \wedge \quad \neg q.j.k \;\Rightarrow\; (\lhd\!\!\square : age.j.k * S : \neg q.j.k))
\end{array}
$$

The first conjunct of $\mathcal{TI}$ asserts that if node k is up, all of its neighbors j will receive k's information within KS time. The second conjunct asserts the correctness of $age.j.k$: $age.j.k$ is only reset upon the receipt of information; if $age.j.k = n$, either no information has been received in the previous n rounds or some information has been received but not processed. In other words, if $age.j.k = n$ and no information has been received, that has not been processed (as indicated by $q.j.k$ holding), then it must be true that no message has been received in the last n rounds which lasted at least nS time.

State Invariant:

$$
\begin{array}{lll}
\mathcal{SI} & \equiv & (\forall j,k : up.j : \\
& & \quad (adj.j.k \;\equiv\; up.k \wedge up.\langle k,j\rangle) \\
& & \wedge \quad ((\neg up.k \;\vee\; \neg up.\langle k,j\rangle) \;\Rightarrow\; \neg q.j.k) \\
& & \wedge \quad (\neg adj.j.k \;\Rightarrow\; age.j.k = K))
\end{array}
$$

Stabilization: No matter how the network topology or the protocol state is perturbed, continued execution of our protocol will eventually yield a legal state. The worst case stabilization time of the protocol is $(K+1)T + R$.

Due to lack of space, we refer the reader to [12] for all proofs of timing invariants, state invariants, and stabilization.

3.2 Extensions

Tolerating Clock Drift: Thus far we have assumed that all node clocks measure real-time ideally, and hence each node can accurately schedule its iteratively executed action in the interval $[S, T)$. In this subsection, we show how to extend our solution to tolerate node clocks that do not measure real-time ideally, as long as the cumulative error between any two node clocks is at most Δ, where Δ is a known constant.

We observe that it is possible for the timing constraint $R+T < KS$ to hold for ideal clocks, but be violated when clocks drift apart by at most Δ. In particular, a node with a slow clock may require up to $T + \Delta$ to complete one iteration. If $R + T + \Delta \geq KS$, successive keep-alive messages from the node with the slow clock may not arrive at the node's neighbors before the neighbors mark the node down.

To solve this problem we add a constant delay, c, to the lower time bound of the node action. The value of c is chosen so that

$$\frac{R + T + \Delta - KS}{K} < c$$

Since it is known that $R+T < KS$, the value of c yields the more restrictive timing constraint,

$$R + T < K(S + c)$$

This new constraint allows local clocks to drift from each other by at most Δ while preserving the proofs of correctness (with S replaced by $S + c$).

Accounting for clock drift, the node action for node j becomes:

$up.j \xrightarrow{[S+c,T)}$
$(\| \; k : \langle k, j\rangle :$
$\quad ch.j.k := ch.j.k \;\cdot\; ?$
$\;;$ **if**
$\quad q.j.k \;\rightarrow\; adj.j.k, age.j.k, q.j.k := true, 0, false$
$\quad [\!]$
$\quad \neg q.j.k \wedge age.j.k < K \;\rightarrow\; age.j.k := age.j.k + 1$
$\quad [\!]$
$\quad \neg q.j.k \wedge age.j.k = K \;\rightarrow\; adj.j.k := false$
fi)

Interrupt Version: The adjacency protocol given previously is based on "polling": node j iteratively polls each of the $q.j.k$ to determine on which channels new information had been received. Based on this polling, the adjacency relation is maintained.

For certain network architectures, it may be appropriate to design a stabilizing, interrupt driven version of the adjacency protocol: An interrupt version of the adjacency protocol is readily designed that differs from the polling version in that it modifies the adjacency relation as soon as it receives new information.

4 A Stabilizing Timing-Based Connectivity Protocol

Specification: It is required to design for each node j a set of actions that maintain for each node k a variable $conn.j.k$, for k is connected to j, such that $conn.j.k$ is true iff j can prove there exists a path from node j to node k consisting only of up channels and up nodes.

Design: While the type of information exchanged by the nodes in connectivity is the same as in adjacency, in connectivity the information must be propagated to all nodes in the network. We propagate the information by "piggybacking" local state on top of the keep-alive messages.

In order for information to be retracted in every node, nodes must have some method of propagating the retraction. We add some information to the local state of each node, namely its alleged distance from every other up node, to facilitate this retraction. A node may retract information by "retracting" its distance value (by assigning an impossible distance, such as N). A vector, $dist.j$, contains the alleged distances from node j to every other node in the network.

Remark: We will occasionally subscript variables to emphasize the fact that they are local variables (or local copies of variables). The subscript denotes which node owns variable; thus, $conn_j.k$ denotes node j's local copy of node k's connectivity vector. (End of Remark)

The node action for node j is:

$up.j \xrightarrow{[S,T)}$
$\quad (\|k : \langle j, k\rangle : ch.j.k := ch.j.k \;\cdot\; dist_j.j)$
$;\; dist_j.j.j := 0;\;\; (\|i : i \neq j : dist_j.j.i := N)$
$;\; (\|k ::$
$\quad$ **if**
$\quad\quad (\exists i : \langle i, j\rangle : dist_j.i.k < N) \;\rightarrow$
$\quad\quad\; age.j.k, conn.j.k, dist_j.j.k := 0, true, dist_j.i.k + 1$
$\quad [\!]$
$\quad\quad (\forall i : \langle i, j\rangle : dist_j.i.k = N) \;\wedge\; age.j.k < K \;\rightarrow$
$\quad\quad\; age.j.k := age.j.k + 1$
$\quad [\!]$
$\quad\quad (\forall i : \langle i, j\rangle : dist_j.i.k = N) \;\wedge\; age.j.k = K \;\rightarrow$
$\quad\quad\; conn.j.k := false$
$\quad$ **fi**)

These actions are similar to the actions in the adjacency protocol, the main difference being the additional proof, a legal $dist$ value, needed to verify the correctness of the received information.

A problem could arise if false $dist$ values were allowed to persist in the network, as a false $dist$ value could prevent nodes from retracting information; therefore, we must show that no incorrect $dist$ values persist in the network.

In each round, every node resets its $dist$ values to N. Furthermore, $dist$ values less than N that are received from neighbors are always incremented. Thus, whereas an up node j will periodically rebroadcast a $dist_j.j.j = 0$ to prevent others from marking it down, a node j that is down will not broadcast any $dist_j.j.j$ value, so all other nodes, k, will continually increase their $dist_k.k.j$ until all $dist_k.k.j$ become N, thereby enabling others to mark it down.

4.1 Verification

Correctness of the connectivity protocol is proven in the same manner as the correctness for the adjacency protocol: we exhibit a $\mathcal{TI}$ which is used to prove $\mathcal{SI}$ is closed. We also assume the same timing constraint as in the adjacency protocol, namely $R + T < KS$.

Timing Invariant:

$$\begin{array}{l} \mathcal{TI} \;\equiv\; (\forall i,j,k : up.j : \\ \qquad (conn.k.i \;\wedge\; up.k \;\wedge\; up.\langle k,j\rangle) \;\Rightarrow \\ \qquad\quad (\rhd : KS : conn.j.i) \\ \quad \wedge\; \neg conn.j.i \;\Rightarrow\; (\lhd\Box : age.j.k * S : \neg conn.j.i)) \end{array}$$

Again, similar to the adjacency protocol, the first conjunct $\mathcal{TI}$ states that each node will propagate its *conn* values (via its *dist* values) in a timely manner, and the second conjunct states that $age.j.k$ correctly indicates the minimum number of iterations since node j received verification of i's reachability.

State Invariant:

$$\begin{array}{l} \mathcal{SI} \;\equiv\; (\forall j,k : up.j : \\ \qquad (conn.j.k \;\equiv\; reach.j.k) \\ \quad \wedge\; ((\exists i : \langle i,j\rangle : dist_j.i.k < N) \;\Rightarrow\; reach.j.k) \\ \quad \wedge\; (reach.j.k \;\Rightarrow\; (age.j.k < K \;\vee \\ \qquad\qquad (\exists i : \langle i,j\rangle : dist_j.i.k < N))) \\ \quad \wedge\; (\neg conn.j.k \;\Rightarrow\; age.j.k = K)) \end{array}$$

The first conjunct ensures the first condition of the specification is met: *conn.j.k* is true iff node k is reachable from node j in the physical network (denoted $reach.j.k$). The next two conjuncts ensure that information is correctly propagated: first, if node j has received $dist_j.i.k < N$ from a neighbor i, then node k must be reachable from node j, and second, if k is reachable from j, then j will receive proof of the fact before $age.j.k = K$. Finally, the last conjunct simply states that *conn.j.k* can not be falsified before $age.j.k = K$.

Stabilization: The total stabilization time for the connectivity protocol is $(K+d)T+dR$. We note the maximum stabilization time is roughly equivalent to the maximum end-to-end transmission time of the network which makes the protocol good for use in high speed networks or when low delay is crucial.

Extensions: The connectivity protocol can be extended to accommodate clock drift and interrupts in the same manner the adjacency protocol was extended to accommodate them.

5 A Stabilizing Timing-Based Protocol Schema for Information Exchange

Specification: Given a global relation, $rel.j.k$, between the nodes of the network, it is required to design for each node j a set of actions that maintain for each node k a boolean variable $assert.j.k$, for node j asserts $rel.j.k$ holds, such that $assert.j.k$ is true iff j can prove that $rel.j.k$ holds.

Design: Using the connectivity problem as a basis, we present a protocol schema for solving all information exchange problems. The connectivity problem requires the propagation of information to all reachable nodes. By specifying what information is exchanged and how the information is used, we can use the same schema to solve many different problems. For example, the Leader Election problem can be solved by having each node "elect" the connected node with the highest id as the leader (e.g., $leader.j = \max\{k \,|\, conn.j.k\}$); or, if each node notes which neighbor first provided the proof of *conn.j.*(*leader.j*), the Leader Election solution can be extended to Spanning Tree Construction; or, finally, if instead of ids, the nodes exchange arbitrary numbers, the Leader Election protocol becomes a protocol for solving the Maxima Finding problem.

In this section we present a stabilizing, timing-based protocol schema for Information Exchange. In places where problem-specific information is required we use placeholders — the information that is exchanged and how the information is used is varied by varying the instantiations of the placeholders.

5.1 Schema Placeholders

The placeholders include the data structure *st.j*, the local state of j; *q.j.k*, a temporary buffer for received information; and *pst.j.k*, a copy of the last information node j received from node k: *pst.j.k* is node j's local copy of *st.k*. Because *q.j.k* is only used as a buffer, and it has the same structure as *st.j*, we will not explicitly specify *q.j.k*.

In addition to the data structures, we specify a boolean expression defined over *pst.j.j* and *pst.j.i*, *proof.j.i.k*, that is true only if j has current information from some node i that proves $rel.j.k$ holds — *proof.j.i.k* denotes the states in which $assert.j.k$ can be truthified. For example, in the connectivity protocol if node j receives a $dist_j.i.k < N$ then node j assert that node k is reachable; thus, $dist_j.i.k < N$ serves as a proof that node k is reachable from node j.

Finally, we also specify two statements. The first statement, **reset**, performs a reset on *st.j*. When nodes maintain information asserted by nodes other than their neighbors, they must rely on a third party to provide valid information. If unchecked, false information may persist in the network, so nodes require all information received from a third party be verified. Since the third party cannot get verification for the false information, the third party will eventually be forced to retract the information. In order to force a verification check each time information is received from a third party, a node resets some pieces of *st.j* each iteration to some known illegal value. If the information is verified, the node will send legal values to its neighbors; if not, the node will send the known illegal value to its neighbors so the neighbors will know the information is false.

The second statement, **update**, updates *st.j* with information from *pst.j.k*. The **update** statement is often used to maintain the auxiliary information used to verify the information received from third parties.

5.2 Schema Overview

The structure of the schema's node action is similar to the structure of the node action in the connectivity protocol, but the schema requires that additional operations be performed.

As before, node j first sends its local state to all its neighbors. Node j then checks for information received since the last iteration (denoted by a non-empty $q.j.k$). If node j has received information from node k, it makes a copy of the temporary buffer, $q.j.k$, where the information was stored, and clears $q.j.k$ to indicate there is no new information from node k. Since node j may modify its local information, and therefore erase a proof, during a round, node j makes a stable copy of its local information: this copy is the information known to node j at the end of the previous iteration (as it has not been modified in this iteration). Finally, the last overhead statement resets $st.j$.

The final statement of the node action, the parallel conditional statement, is similar to the final statement of the connectivity problem, but the specifics of the connectivity protocol have been replaced with placeholders. As in the connectivity protocol, the relation being maintained by the schema is updated in one of three ways:

1. If node j received proof in the previous round from some neighbor i that $rel.j.k$ holds, we note node j has received a current proof of $rel.j.k$ by truthifying $assert.j.k$ and resetting $age.j.k$. Furthermore, $st.j.k$ is updated to reflect any additional information received from node i concerning node k.

2. If node j did not receive any proof in the previous round that $rel.j.k$ holds, we "age" j's local copy of k's information; that is, we say k's information at node j is less current than it was the previous round.

3. If node j has not received any proof in any of the previous K rounds that $rel.j.k$ holds, either node k has failed or there is no path of up nodes and channels from node k to node j; therefore, node j cannot receive information from node k, so the information is retracted.

The node action for node j is:

$$
\begin{array}{l}
up.j \xrightarrow{[S,T)} \\
\quad (\|k : up.\langle j,k\rangle : ch.j.k := ch.j.k \;\cdot\; st.j) \\
;\ pst.j.j := st.j;\ \ (\|i : i \neq j : pst.j.i, q.j.i := q.j.i, \langle\rangle) \\
;\ \textbf{reset}\ st.j \\
;\ (\|k :: \\
\quad \textbf{if} \\
\qquad (\exists i :: proof.j.i.k) \ \rightarrow\ age.j.k, assert.j.k := 0, true \\
\qquad\qquad\qquad\qquad\qquad ;\ \textbf{update}\ st.j.k \\
\quad \square \\
\qquad (\forall i :: \neg proof.j.i.k) \ \rightarrow\ age.j.k := age.j.k + 1 \\
\qquad\quad \wedge\ age.j.k < K \\
\quad \square \\
\qquad (\forall i :: \neg proof.j.i.k) \ \rightarrow\ assert.j.k := false \\
\qquad\quad \wedge\ age.j.k = K \\
\quad \textbf{fi})
\end{array}
$$

5.3 Verification

Just as the protocol schema permits us to quickly develop information exchange protocols, we present a proof schema based on timing and state invariants to quickly prove the correctness of the resulting protocols. We assume the same timing constraint as in the connectivity protocol, namely $R+T < KS$. While the timing invariant will remain unchanged for all information exchange protocols, the state invariant is protocol specific.

Timing Invariant:

$$
\begin{array}{l}
\mathcal{TI} \ \equiv\ (\forall j,k : up.j : \\
\qquad (up.k \ \wedge\ up.\langle k,j\rangle) \ \Rightarrow\ (\rhd : KS : q.j.k) \\
\quad \wedge\ \neg(\exists i :: proof.j.i.k) \ \Rightarrow \\
\qquad\quad (\lhd\square : age.j.k * S : \neg(\exists i :: proof.j.i.k)))
\end{array}
$$

State Invariant:

$$
\begin{array}{l}
\mathcal{SI} \ \equiv\ (\forall j,k : up.j : \\
\qquad (assert.j.k \ \equiv\ rel.j.k) \\
\quad \wedge\ ((\exists i :: proof.j.i.k) \ \Rightarrow\ rel.j.k) \\
\quad \wedge\ (rel.j.k \ \Rightarrow\ (age.j.k < K \ \vee \\
\qquad\qquad\qquad (\exists i :: proof.j.i.k))) \\
\quad \wedge\ (\neg assert.j.k \ \Rightarrow\ age.j.k = K))
\end{array}
$$

The first conjunct of $\mathcal{SI}$ states that the protocol variable $assert.j.k$ is true iff the desired relation, $rel.j.k$, does indeed hold between nodes j and k; that is, the protocol meets its specification by correctly maintaining the desired relation. The next two conjuncts ensure that no false proofs exist and that if the relation does hold, it can be proven: if a proof exists, the relation holds; and if the relation holds, a proof is received within K rounds. The last conjunct of $\mathcal{TI}$ is the correctness condition for $age.j.k$ as it relates to $assert.j.k$ — $\neg assert.j.k$ can hold only if no proof has been received for K consecutive rounds.

Stabilization: Like the connectivity protocol, the protocol schema stabilizes in at most $(K+d)T + dR$ time. We note the best case end-to-end transmission time is 0 and the worst case end-to-end transmission time is $d(T+R)$.

Extensions: Again, protocol extensions analogous to the clock drift and interrupt processing extensions given for the adjacency protocol may be applied to this protocol schema.

6 Adaptivity of the Schema

We show in this section that our protocol schema is readily adapted to suit a variety of network loads, delay requirements, and scheduler restrictions, by exploiting the degrees of freedom allowed by its timing constraints.

More precisely, we observe that the constraints $T+R < KS$ and $S < T$ admit the following degrees of freedom:

1. T can be increased as long as $T+R < KS$:
By increasing T, the interval $[S,T)$ is lengthened which makes the scheduling of the node action easier in heavily loaded nodes.

2. S can be decreased as long as $T+R < KS$:
Decreasing S also lengthens the interval $[S,T)$ making scheduling easier.

3. T can be reduced as long as $S < T$:
Since the worst case end-to-end delay is $d(T+R)$, where d is the diameter, as T decreases the end-to-end delay decreases. Similarly, the best case delay is 0, yielding an end-to-end variance of $d(T+R)$; thus, reducing T also decreases the end-to-end variance in delay. Related to the end-to-end delay is the stabilization time, $(K+d)T + dR$; again, as T decreases, the stabilization time decreases.

4. S can be increased as long as $S < T$:
Recall that tolerance to clock drift was enabled by

adding a constant to S when implementing the node action; thus increases in S provide greater tolerance to clock drift.

5. K can be decreased as long as $T + R < KS$:
   Again, the stabilization time is $(K + d)T + dR$, so decreasing K decreases the stabilization time.
6. K can be increased freely :
   Increasing K provides greater tolerance to transient failures, such as message loss or corruption, since short-lived absence of correct information will not suffice for nodes to retract information.

We further observe that our decision to use the same R for all channels and the same S, T, and K for all nodes is over-specific. In fact, more generally, we could have let $S.j$ and $T.j$, $S.j < T.j$, be unique lower and upper timebounds on node action of j, $K.j$ be the maximum age used in the node action of j, and $R.\langle j, k\rangle$ be the upper bound on the communication delay on the channel from j to k and shown that the schema is correct provided the following timing constraint is satisfied $(\forall \mathbf{j}, \mathbf{k} : \mathbf{R}.\langle \mathbf{k}, \mathbf{j}\rangle + \mathbf{T}.\mathbf{k} < \mathbf{K}.\mathbf{j} * \mathbf{S}.\mathbf{j})$. It now follows that different degrees of freedom may be applied (and to differing degrees) at the various nodes, depending upon the number of neighboring nodes, the load on each node, the congestion on neighboring channels, etc.

We finally observe that changes in the values of $S.j$, $T.j$, and $K.j$ can be made online, as long as the timing constraints are preserved in the presence of changes.

7 Conclusions

We have systematically developed both a uniform protocol schema for stabilizing information exchange in computer networks and a uniform verification schema to verify the correctness of the resulting protocols.

Our schema is based on well-established concepts. The protocol notation uses a modified form of Dijkstra's guarded commands [16]; the use of upper and lower time bounds on statement execution is based on earlier work [9, 15, 17]; protocols are verified by exhibiting a state invariant and a timing invariant; and, timing invariants are proven using a small number of well understood temporal concepts [15].

Our schema accommodates the exchange of both fixed and time-varying information. While protocols for both types of information are developed in the same manner, their implementation is potentially different. For instance, in the latter case, the use of a smaller upperbound T may be crucial for ensuring small message sizes and low transmission delay.

The schema is designed using timing-based actions. The use of timing-based actions obviated the need for explicit acknowledgements of receipt of information, which facilitates the development of continuous-media data exchange protocols in high speed networks.

With respect to implementation, our schema is tolerant to two common timing problems: the absence of (hard) guarantees that actions can be scheduled with exact periodicity, and imprecise local clocks. The timebounds for its actions can be adapted in various ways, even dynamically, to improve performance, facilitate schedulability, accommodate varying network traffic, etc. The overhead it imposes on the size of the information messages is at most a single vector of N distance values.

Finally, at the moment we have only ensured the protocols are stabilizing but not that they are masking: during the convergence phase, nodes may have inconsistent information. While short periods of inconsistency may be acceptable for some applications, such as video transmission, for some applications such as navigation systems, even short periods of inconsistency can be fatal. We are currently extending our schema so that it masking, in addition to being stabilizing.

Acknowledgements. We thank the referees for their constructive comments.

References

[1] M. G. Gouda and N. Multari. Stabilizing communication protocols. *IEEE Transaction on Computers*, 40(4):448–458, 1991.

[2] G. Varghese. *Self-Stabilization by Local Checking and Correction*. PhD thesis, Massachusetts Institute of Technology, 1992.

[3] A. Arora and D. M. Poduska. A logical foundation of real-time programs. Unpublished Manuscript, 1994.

[4] E. A. Ashcroft. Proving assertions about parallel programs. *Journal of Computer and System Sciences*, 10:110–135, 1975.

[5] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11), 1974.

[6] D. Gries. *The Science of Programming*. Springer-Verlag, New York, 1981.

[7] F. Jahanian and A. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transaction on Software Engineering*, 12(9):890–904, 1986.

[8] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. Technical Report STAN-CS-90-1307, Stanford University, 1990.

[9] N. A. Lynch and H. Attiya. Using mappings to prove timing properties. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 265–280, Quebec City, PQ CDN, 1990. ACM Press, New York, NY, USA.

[10] N. A. Lynch and F. Vaandrager. Forward and backward simulations – part ii: Timing-based systems. *Information and Computation*, 128(1):1–25, 1996.

[11] R. Koymans, J. Vytopil, and W. P. de Roever. Real-time programming and asynchronous message passing. In *2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 1983.

[12] A. Arora and D. M. Poduska. A timing-based schema for stabilizing information exchange. Technical Report OSU-CISRC-5/95-TR26, The Ohio State University, 1995.

[13] A. Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, IT-29:23–35, 1983.

[14] E. W. Dijkstra. *A Discipline of Programming*, chapter 14. Prentice-Hall, Englewood Cliffs, N. J., 1976.

[15] T. A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proceedings of the 18th Annual Symposium on Principles of Programming Languages*, pages 353–366, Orlando, Florida, January 1991. ACM Press.

[16] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[17] F. Modugno, M. Merritt, and M. Tuttle. Time constrained automata. In *CONCUR '91 Proceedings of Workshop on Theories of Concurrency*, Amsterdam, August 1991.

8 Appendix: Applying the Schema

Our schema enables quick and easy development of protocols for information exchange problems. In this appendix, we illustrate the power of the schema by simply providing instantiations for the placeholders in the schema to solve a number of information exchange problems.

8.1 Connectivity Problem (revisited)

$$\begin{array}{rcl} rel.j.k & \equiv & j = k \ \lor \\ & & (\exists i :: up.i \ \land \ up.\langle i,j\rangle \ \land \ rel.i.k) \\ st.j & \equiv & (dist_j.j) \\ \textbf{reset} & \equiv & dist_j.j.j := 0 \\ & & ; \ (\|i : i \neq j : dist_j.j.i := N) \\ proof.j.i.k & \equiv & dist_j.i.k < N \\ \textbf{update} & \equiv & dist_j.j.k := dist_j.i.k + 1 \end{array}$$

Since the schema is an extension of the connectivity program, the solution we produce by instantiating the schema is the same as the solution given in Section 4.

Note: We assume that $proof.j.i.k$ holds only when $pst.j.i \neq \langle\rangle$. If $pst.j.i = \langle\rangle$, node j could not have received $dist_j.i$ in the previous round; thus, $dist_j.i$ must be old information. (End of Note)

Note: In the remaining examples we will assume $rel.j.k$ refers only to connected nodes as defined by $rel.j.k$ in this example.

8.2 Commitment Problem

Specification. Each node casts one of two votes, Yes or No, and subsequently reaches one of two decisions, Commit or Abort. It is required that a Commit decision be reached iff all up nodes vote Yes.

$$\begin{array}{rcl} rel.j.k & \equiv & vote_j.j.k = vote_k.k.k \\ st.j & \equiv & (vote.j, dist_j.j) \\ \textbf{reset} & \equiv & dist_j.j.j := 0 \\ & & ; \ (\|i : i \neq j : dist_j.j.i := N) \\ proof.j.i.k & \equiv & dist_j.i.k < N \\ \textbf{update} & \equiv & vote_j.j.k := vote_j.i.k \\ & & ; \ dist_j.j.k := dist_j.i.k + 1 \end{array}$$

Each node maintains an array, $vote_j.j$, of the votes it has received. If every up node votes Yes — $vote_j.j.k$ holds for all nodes k reachable from j — the node reaches the decision of Commit. We assume $vote_j.j.j$ is determined externally by node j.

8.3 Leader Election Problem

Specification. It is required that each up node know the id of the up node whose id is the highest of the up nodes.

$$\begin{array}{rcl} rel.j.k & \equiv & j = k \ \lor \\ & & (\exists i :: up.i \ \land \ up.\langle i,j\rangle \ \land \ rel.i.k) \\ st.j & \equiv & (dist_j.j) \\ \textbf{reset} & \equiv & dist_j.j.j := 0 \\ & & ; \ (\|k : k \neq j : dist_j.j.k := N) \\ proof.j.i.k & \equiv & dist_j.i.k < N \\ \textbf{update} & \equiv & dist_j.j.k := dist_j.i.k + 1 \end{array}$$

Our solution to the leader election problem is almost the same as our solution to the connectivity problem, the only difference being that a "leader" — the reachable node with the highest id — is chosen, $leader_j.j := \min\{j | assert.j.k\}$: if node k is reachable from node j, we know from the connectivity protocol that both nodes will have identical $assert$ vectors and, therefore, choose the same leader.

8.4 Maxima Finding Problem

Specification. Each node asserts a value m. It is required that each node maintain a variable mf whose value is the maximum m value of the up nodes.

$$\begin{array}{rcl} rel.j.k & \equiv & (\forall i :: m_j.i = m_k.i) \\ st.j & \equiv & (dist.j, m.j) \\ \textbf{reset} & \equiv & dist.j.j := 0 \\ & & ; \ (\|k : k \neq j : dist.j.k := N) \\ proof.j.i.k & \equiv & dist_j.i.k < N \\ \textbf{update} & \equiv & dist.j.k := dist_j.i.k + 1 \end{array}$$

Our solution to the maxima finding problem is similar to the solution to the leader election problem except that instead of choosing the highest id of the reachable nodes as a leader, the maximum m value is chosen, $mf_j.j := \max\{m_j.k | assert.j.k\}$.

8.5 Spanning Tree Construction

Specification. It is required to maintain a tree that spans all up nodes.

$$\begin{array}{rcl} rel.j.k & \equiv & j = k \ \lor \\ & & (\exists i :: up.i \ \land \ up.\langle i,j\rangle \ \land \ rel.i.k) \\ st.j & \equiv & (dist_j.j) \\ \textbf{reset} & \equiv & dist_j.j.j := 0 \\ & & ; \ (\|i : i \neq j : dist_j.j.i := N) \\ proof.j.i.k & \equiv & dist_j.i.k < N \\ \textbf{update} & \equiv & dist_j.j.k := dist_j.i.k + 1; \\ & & \neg assert.j.(father.j) \rightarrow father.j := i \end{array}$$

Our solution to the problem is an augmentation of our solution to the leader election problem: each node has a "father" that may be updated only when the node executes its update statement. The graph of the father relation defines the desired spanning tree.

8.6 Routing Problem

Specification. Each channel $\langle j,k\rangle$ has an associated cost, $d.j.k$. It is required that each node maintain a route of minimal cost from itself to every other up node in the network.

$$\begin{array}{rcl}
rel.j.k & \equiv & (father.j.k = i) \;\Rightarrow \\
 & & (\forall l,m :: d.j.i{+}v.i.k \le d.j.l{+}v.l.m) \\
st.j & \equiv & (v.j, d.j) \\
\textbf{reset} & \equiv & v_j.j.j, father.j.j := 0, j \\
 & & ;\;\; (\|i : i \neq j : v_j.j.i := +\infty) \\
proof.j.i.k & \equiv & d_j.j.i + v_j.i.k < v_j.j.k \\
\textbf{update} & \equiv & v_j.j.k := d_j.j.i + v_j.i.k + 1 \\
 & & ;\;\; father.j.k := i
\end{array}$$

A number of trees are created, one tree rooted at each node, such that the distance from each root to each node in its tree is minimal. The accumulated distance between nodes k and j is maintained in variable $v_j.j.k$. Each tree is represented by the father relation, $father.j.k$ being j's father in the tree rooted at k.

8.7 Clock Synchronization

Specification. Each node maintains a nondecreasing physical clock $pc.j$. It is required that each node maintain a variable $c.j$ such that $(\forall j,k :: |c_j.j - c_k.k| < \delta)$, for some δ.

$$\begin{array}{rcl}
rel.j.k & \equiv & assert.j.k \;\Rightarrow \\
 & & 0 \le pc_k.k{-}pc_j.k \le dist_j.j.k{*}(T{+}R) \\
st.j & \equiv & (dist.j, pc.j) \\
\textbf{reset} & \equiv & dist_j.j.j := 0 \\
 & & ;\;\; (\|k : k \neq j : dist_j.j.k := N) \\
proof.j.i.k & \equiv & dist_j.i.k < N \\
\textbf{update} & \equiv & dist_j.j.k := dist_j.i.k + 1 \\
 & & ;\;\; vc_j.k := vc_i.k
\end{array}$$

It is often necessary for network nodes to maintain a common notion of time. One method of accomplishing this goal is for each node to maintain a logical clock based on the hardware clocks of the nodes in the system. One example would be $c.j = \min\{pc_j.k | assert.j.k\}$. The maximum distance between any two nodes is the diameter of the network; therefore, the accuracy of the logical clocks is $diameter * (T + R)$.

The clock synchronization problem is a classic problem using continuous information. The physical clocks of each node constantly change, yet we are able to design a stabilizing protocol for this problem just as easily as we did for discrete information problems.