Proceedings of the 1984
Winter Simulation Conference
S. Sheppard, U. Pooch, D. Pegden (eds.)



# PROBLEM ORIENTED PROTOCOL DESIGN

*David M. Nicol and Paul F. Reynolds, Jr.*
*Department of Computer Science*
*University of Virginia*
*Charlottesville, Virginia 22901*

**Abstract**

The prevention of deadlock in certain types of distributed simulation systems requires special synchronization protocols. These protocols often create an excessive amount of performance-degrading communication; yet a protocol with the minimum amount of communication may not lead to the fastest network finishing time. We propose a protocol that attempts to balance the network's need for auxiliary synchronization information with the cost of providing that information. Using an empirical study, we demonstrate the efficiency of this protocol. Also, we show that the synchronization requirements at different interfaces may vary; an integral part of our proposal assigns a protocol to an interface according to the interface's synchronization needs.

## 1. Introduction

Many physical systems that we simulate are inherently parallel; the time required to simulate these systems can often be significantly reduced by performing the simulation on a distributed network. A major class of distributed architectures prohibits the efficient implementation of a global simulation clock; each processor maintains its own simulation clock. Careful synchronization of processors in such a network is required to ensure the correctness of the simulation results; further care is required to ensure that the network does not deadlock as a result of the synchronization. A number of deadlock-free synchronization protocols have been proposed, but many of these suffer from excessive message traffic. The SRADS[1] protocol uses comparatively fewer messages, but suffers from time inaccuracy: an inaccurate correspondence between simulation time and the modeled real time. Using SRADS as a base, we present a protocol that prevents deadlock, avoids time inaccuracy and attempts to minimize the non-essential message traffic.

We outline an overview of this paper. Section 2 describes our model of distributed simulation networks, and lists criteria we ask distributed simulation systems to satisfy. We briefly sketch how some synchronization protocols work; we observe that minimizing communication costs need not minimize network completion time. In section 3 we look at the SRADS synchronization protocol, and see that SRADS allows time inaccuracy. Section 4 introduces the appointment. We present the results of an empirical study indicating the appointment's efficiency. We show that the appointment corrects SRADS' time inaccuracy and still prevents deadlock. Our final section shows how an information interface may be relatively insensitive to the values of message time-stamps. We argue that the SRADS protocol may be used at such interfaces while using the appointment elsewhere.

## 2. Distributed Simulation Networks

We describe here our model of distributed simulation networks. We assume that the physical system being simulated is composed of certain communicating **physical processes**, or **PPs**. **Logical processes** (**LPs**) are software entities isomorphic to PPs; the LPs simulate the desired characteristics of PPs. LPs simulate the passage of messages between PPs by sending **content messages**. In order to synchronize properly, the LPs also exchange **protocol messages**. A protocol message does not model any message in the simulated physical system. An LP's simulation time is maintained in its own **logical clock**, generally denoted by $C$. Each content message an LP generates is stamped with the LP's logical time, and is conceptually written into a **shared facility** (**SF**). An SF may have several readers and writers. A content message written to an SF is physically sent to each reader of that SF. The LPs are executed on a network of processors that share no memory: all communication is message based. Each LP runs on its own processor.

Given that the distributed simulation of a system is a viable option, we may want the running simulation to satisfy the following criteria.

(1) The network does not deadlock.

(2) If $PP_i$ communicates with $PP_j$ at time $n$ then $LP_i$ sends a content message to $LP_j$ at logical time $n$.

(3) The sequence of time stamps on content messages that cross an interface is monotonically increasing.

The first criterion is clearly desirable, for if we cannot prevent deadlock we must detect and correct it. The second demands that a content message's logical time stamp accurately models the *physical* time of the simulated *physical* message; we will call this the **time accuracy** criterion. The last criterion requires that simulation messages crossing an interface arrive at an LP in the same order as their corresponding physical messages arrive at a PP. We call this the **correctness** criterion. The fulfillment of these criteria is achieved through the use of synchronization protocols; when an LP wants to send or process a content message, it may have to wait until the system satisfies certain conditions. Several protocols have been proposed for this purpose. We illustrate by example the general philosophy of the Link Time[2] and Null Message[3] protocols.

Consider the queueing network shown in Figure 1, noting that that each server and queue is simulated by its own LP.

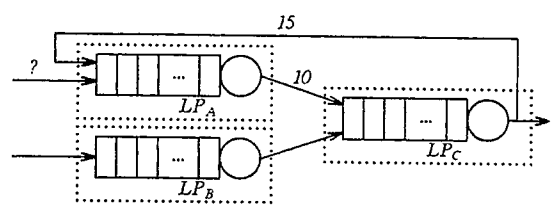



*Figure 1: Distributed Simulation of a Queueing Network*

$LP_A$ may receive jobs both from $LP_C$ and an external source; it must consequently take some care to enforce the correctness criterion on its interface with $LP_C$. Suppose $LP_A$'s clock has value 10, and a job with time stamp 15 arrives from $LP_C$. The time of the next externally generated job is not known. $LP_A$ may not be able to advance its clock to 15 and service the new job, for suppose it does. An externally generated job with time stamp 14 might later arrive; under FCFS service this job should receive service first. Its

*This work was supported by a grant from the IBM corporation.*

logical finishing time (and time stamp on the subsequent content message to $LP_C$) is less than the finishing time of the job arriving with time 15. Yet the latter job's content message is sent to $LP_C$ first, violating the correctness criterion.

Both the Link Time and Null Message models handle this dilemma by associating a logical time with each LP interface. This time represents a lower bound on the time stamp of the next content message to cross that interface. To ensure correctness, an LP may not simulate past its minimum received bound. Each interface's writing LP is solely responsible for updating this bound. The prevention of deadlock is assured by the assumption of a minimum difference $\epsilon > 0$ between the time stamps of consecutive content messages across an interface. In situations similar to that depicted by Figure 1, these lower bounds may most often be incremented only by $\epsilon$. Since passing a bound requires the generation of a protocol message, computational and communication costs are associated with each such increment. For small $\epsilon$, the majority of the network's messages are protocol messages. The cost of preventing deadlock in such a network can be quite high.

Excessive communication can lead to degraded performance, but minimizing communication need not optimize performance. The network in Figure 1 can illustrate this as well. Suppose that the only bounds on message times provided by $LP_A$ to $LP_C$ are the time stamps on content messages. Furthermore suppose that $LP_A$'s external source provides it with protocol messages establishing lower bounds on future content message times. These bounds allow $LP_A$ to calculate bounds on its own future content message times. If $LP_A$ does not send $LP_C$ protocol messages establishing these bounds, $LP_C$ can never process a job with a time stamp less than the time of the last content message received from $LP_A$. An arbitrarily large queue of messages from $LP_B$ can build up in $LP_C$. Some of these queued messages might be processed if $LP_A$ would pass lower bounds to $LP_C$. Without these bounds, maximum possible concurrency is not being exploited, leading to a likely increase in network completion time.

The SRADS protocol prevents deadlock with limited communication; however, SRADS does not satisfy the time accuracy criterion. Our proposed protocol attempts to retain the message efficiency of SRADS while satisfying the time accuracy criterion.

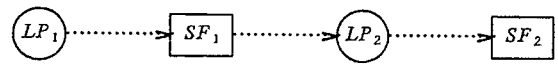
## 3. SRADS and Time Accuracy

This section describes the SRADS protocol, and shows how it allows the violation of the time accuracy criterion. Our understanding of this violation leads to a corrective measure.

The original SRADS protocol was given in [1]. A considerable amount of research effort has gone into optimizing and evaluating SRADS[4-6]; the use of SRADS in simulating logic networks has been studied on an actual distributed system[7,8]. SRADS differs from other protocols most notablely in its use of active readers. In other protocols, readers wait passively for messages to be sent to them. SRADS readers decide for themselves when to look for messages. Since no lower bounds protocol messages are needed, an SRADS distributed simulation can execute correctly with comparatively few protocol messages.

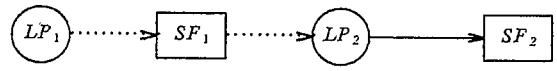
When an LP decides to read an SF, it first waits until all writers of that SF have clock values at least as large as its own. This both enforces the correctness criterion and ensures that the data used is the most current. An SRADS writer may send a message any time so long as buffer space is available for that message. Under these rules, SRADS ensures freedom from deadlock using only finite communication buffer space. However, it may violate the time accuracy criterion, as illustrated by the following example.

Consider a network including a logical process $LP_1$ that generates data for a process $LP_2$. Part of $LP_2$'s task is to monitor its shared facility with $LP_1$. Upon detecting a message from $LP_1$, $LP_2$ accepts the message and calculates a delay time t. We assume that the actual value of t is a function of the data in the message that $LP_1$ sends. Then t logical units after accepting the message from $LP_1$, $LP_2$ sends a message to $SF_2$. $LP_2$ may also independently send other messages to $SF_2$. $SF_2$ is potentially read by any other LP in the network. In SRADS, reading LP's determine when to examine shared facilities for data, so we may suppose that $LP_2$ reads from the $SF_1$ every 10 logical units. The following sequence illustrates how SRADS allows time inaccuracy.



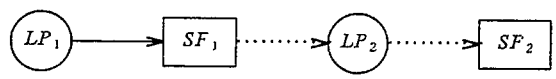

$C_1 = 0$ TS = Ø $C_2 = 0$ TS = Ø

The simulation begins. Both $LP_1$ and $LP_2$ start at time 0.



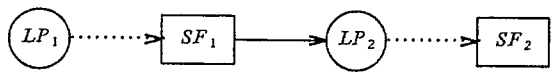

$C_1 = 4$ TS = Ø $C_2 = 7$ TS = 7

At logical time 7, $LP_2$ independently writes a message to $SF_2$.



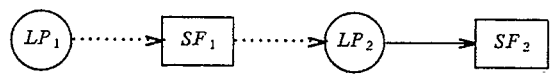

$C_1 = 5$ TS = 5 $C_2 = 9$ TS = 6

At logical time 5 $LP_1$ writes a message to $SF_1$. At the time of the change, $LP_2$ has simulated up to time 9.




$C_1 = 12$ TS = 5 $C_2 = 10$ TS = 6

$LP_2$ reads the changed value in $SF_1$ at logical time 10. It does not block since $C_1 > C_2$.




$C_1 = 13$ TS = 5 $C_2 = 11$ TS = 11

Upon reading the change in $SF_1$ at time 10, $LP_2$ calculates a delay t of 1 logical unit. Thus $LP_2$'s subsequent message to $SF_2$ bears a time stamp of 11.

This example illustrates the effect of allowing $LP_2$ to read a message with a time stamp in its past. In the physical system a transition on $LP_2$'s input line at time 5 would have caused a transition on its output line at time 6. The simulation produces an inaccurate time on $LP_2$'s output line because $LP_2$ does not notice that transition in $SF_1$ until time 10. Even the knowledge that the transition occurred at time 5 doesn't help $LP_2$; the correctness criterion requires that the time stamp on $LP_2$'s next message to $SF_2$ be at least 7.

This example highlights the cause of inaccuracy. $LP_2$ is allowed to advance its clock beyond the simulation time 5, the time at which the transition in $SF_1$ occurred. If $LP_2$ knew that a transition in $SF_1$ might occur at logical time 5, it could block at time 5. Instead, $LP_2$ does not block, accepts a message in its past, and sends a message with an incorrect time stamp. Thus we see that to correct time inaccuracy, we must prevent an LP from receiving a message in its past. The appointment does just that.

## 4. The Appointment

This section introduces the appointment; we argue for the relative efficiency of the appointment and support this argument with empirical evidence. We show that the appointment protocol is free

from deadlock.

An **appointment** is a logical time given by a writer to a reader; the reader agrees not to simulate past this time without the writer's permission. Thus, the appointment ensures that no reader ever receives a message in its past. Stated as such, the appointment appears to be no different than the message time lower bounds discussed in the last section. However, there is a critical difference in when this bound is established.

The models using message time lower bounds require the writing LP to independently maintain the bounds; the interface's reading LP is entirely passive. In SRADS, reading LPs actively query their shared facilities. The appointment adopts the SRADS philosophy, and requires reading LPs to ask for their appointments. An LP never advances its clock beyond the minimum value among all of its appointments; an LP processing a received request for an appointment returns the largest known lower bound on the time of the next message sent across that interface. This bound might be simply its own clock value. It is not immediately clear why the appointment should be any more efficient than the establishment of other models' lower bounds. We can reason heuristically why it might be, and then observe empirically that it is.

### 4.1 A Heuristic Analysis

We can compare the relative workloads of two LPs by comparing their processors' run time utilizations. A lightly utilized LP will tend to advance its clock more quickly while running than a heavily utilized LP; we say that the latter LP is the "slower" of the two. Empirical observations[4] and analytic results[4,9] show that the overall progress of a network is constrained by the progress of the slowest LP.

Consider an interface where the writer is faster than the reader, and the writer independently maintains the bound. Establishing a lower bound extracts a computational cost from the writer. Frequent calculation of the lower bound slows the writer down; furthermore, the slower reader rarely waits at an appointment, the appointment time tends to stay larger than the reader's clock. The writer's extra computation contributes little towards enhancing the slower reader's performance. If the reader requests appointments, the appointments and their associated computational cost occur less frequently. The reader is delayed until it receives the new appointment, but this delay may be offset by the writer's computational gains. The delay can be avoided by requesting the appointment slightly before the last appointment time is reached.

We now reverse the roles, supposing that the writer is slower than the reader; again suppose that the writer independently maintains the bound. The computational costs of establishing a lower bound are even more pronounced since other LPs are often waiting for the writer's results. Reader established appointments can be especially effective in this case. A reader may estimate the rate of the writer's progress, and refrain from requesting an appointment long enough to allow the writer time to accomplish work before having to service an appointment request. A reader's carefully calculated self-imposed blocking need not detrimentally affect overall performance; that reader will usually wait for the writer anyway. We see again that reader requested appointments offer performance gains over writer maintained appointments.

The use of reader requested appointments is thus seen to strike a balance between the network's need for synchronization information and the cost of providing it. The balance is achieved by providing that information only when needed; we have argued intuitively that this balance leads to better network performance. Simulation studies arrived at the same conclusion.

### 4.2 Empirical Results

To evaluate the appointment method, we simulated its use on a number of small (3 to 5 LP) networks. Two LPs in each network share at most one shared facility: each shared facility has exactly one reader and one writer. The physical time elapsed between an LP's data communications is randomly distributed: both positive-normal and exponential distributions are used. We varied a single parameter through each network; this parameter represented writers' abilities to calculate lower bounds on their future message times.

The writers' prediction ability was modeled by associating a logical **write interval** length and a (0,1] real number with each SF an LP writes to. The write interval length is the mean logical time between actual writes that LP performs: the write interval length is constant across all experiments. The (0,1] real is multiplied with the write interval length to produce a shorter interval length, called an **appointment interval**. An LP's logical time line is partitioned into appointment intervals. At the end of an appointment interval, the writing LP randomly decides whether or not to perform a write to the associated SF. The appointment interval length is thus seen to bound a writer's ability to predict the time of its future writes. The write decision is constructed so that the (0,1] real is the probability that the write occurs: for this reason the real is called the write **probability**. The number of logical units between writes is thus seen to be a geometrically distributed random variable with a mean equal to the write interval length. We vary writers' prediction ability by varying the write probability.

We evaluated the appointment protocol by simulating otherwise identical networks with two different methods: the *reader active* and *writer active* methods. Using the writer active method, an LP dispatches an appointment as soon as it reaches the end of an appointment interval. An LP using the reader active method maintains a list of appointment times obtained from its writers. When its clock reaches the minimum value on this list, the LP requests an appointment from the associated writer. An LP processing a request for an appointment can see no farther into its future than the end of its current appointment interval. That endpoint is returned as the appointment time. Every time an LP calculates an appointment message, a computational cost equal to five percent of the mean (real) running time between writes is exacted from it. We assume that the communication delay for the appointment is equal to the computational delay.

To evaluate the effectiveness of reader requested appointments, we compared the simulated performance of several networks using the reader active method with the simulated performance of those same networks using the writer active method. The resulting network completion times are scaled in Figure 2 by the completion times of these networks running with perfect future knowledge. A network using the writer active method with write probabilities equal to 1 yields optimal performance for this model; to say that a network has 50% relative performance is to say that it is twice as slow as that same network running with perfect future knowledge. The graph given in Figure 2 is a composite taken from the networks we tested. It shows the general trend of network performance as we vary the write probability.

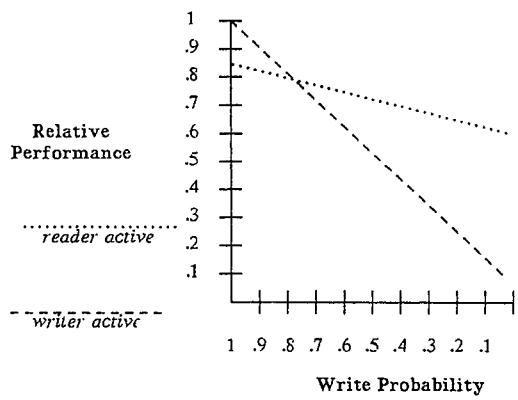



*Figure 2: Relative Performance vs. Write Probability*

These results clearly indicate that the reader active method is less sensitive to a decreasing write probability than is the writer active

method. Furthermore, except for near perfect future knowledge, the reader active method performs better. These results vindicate a use of reader requested appointments.

### 4.3 The Appointment's Freedom from Deadlock

The synchronization behavior of any network using appointments can be simulated by a network using the Link Time model; this model has been shown to be free from deadlock. The only difference lies in how often the critical lower bounds are calculated; the mechanics of establishing appointments does not affect the freedom from deadlock.

## 5. Problem Oriented Protocol Design

We show in this section that an LP can be relatively insensitive to incoming content message time-stamps at some interfaces. It is possible to use the SRADS protocol at such interfaces and still preserve time accuracy on the LP's output messages. We extend this observation to show how we can use a mixture of SRADS and appointment protocols and still ensure the time accuracy of the network.

### 5.1 Time Insensitivity

We have shown how SRADS allows an LP to read a message in its logical past, and consequently post a message with an incorrect time stamp; the LP is sensitive to its received message times. We can conceive of an example where this is not the case.

Consider an SRADS network where $LP_A$ models a logical circuit that at unpredictable intervals toggles an output line read by $LP_E$. $LP_E$ also simulates some logical circuit, but allows itself to be affected by $LP_A$'s output only on the rising edge of some internally generated clock. Assume that the clock has a period of 10 logical $\mu$-secs. The simulation time of $LP_A$'s output is important to $LP_B$ only with a granularity of 10 $\mu$-secs; this is where the SRADS protocol is useful. When $LP_E$ looks for a transition to clock in at time 100. it will recognize the last transition caused by $LP_A$ in the logical interval (90,100]: $LP_E$ is otherwise insensitive to the time of the message. Any content messages that $LP_E$ sends can still have the correct time stamp; thus in special cases we can use the SRADS protocol and still ensure time accuracy.

### 5.2 Tailoring the Protocol to the Interface

The last example showed that an LP may be able to use SRADS at an interface and still maintain the time accuracy of its outgoing content messages. We argue that using SRADS at such an interface can yield better performance. We consider the problems faced by two LPs which can use SRADS at some interfaces but must use appointments at others.

A reader using either the appointment or other discussed protocols at some interface must wait at times established by a writer. If the reader is relatively insensitive at that interface to the times of content messages, it is unreasonable to constrain it from advancing past these times; the reader can determine for itself when it needs to synchronize with its writers. The use of SRADS at such an interface can be more efficient. A single SRADS read accomplishes the needed synchronization; other protocols can require many protocol messages to service the same synchronization need.

Having observed that some interfaces can and should use the SRADS protocol, we consider the problem of mixing SRADS and appointment protocols within a network. We first assume that logical processes and ensuing shared facilities have already been identified. An LP needs to be examined for time sensitivity in each of its input shared facilities. We suppose that some LP's are found to be relatively insensitive to certain of their input shared facilities. Appropriate logical times for that LP to perform SRADS reads on those shared facilities can be calculated. Remembering that a shared facility may have several readers and writers, and that two LPs may share more than one shared facility, we consider the following two questions:

(1) Can two different readers of the same SF use different protocols to read that SF?

(2) Can we use different protocols among all the SFs two LPs may communicate across?

The answer to the first question is yes; a shared facility read by $n$ readers is equivalent to $n$ single reader shared facilities receiving the same messages. The common readers of a shared facility in no way affect each other at that interface.

The answer to the second question is also yes. Suppose that $LP_1$ and $LP_2$ communicate through a number of SFs such that $LP_1$ requires appointments from $LP_2$ at some SFs but can use SRADS at others. So long as $LP_1$ does not advance beyond appointment times without $LP_2$'s permission, the use of SRADS at other interfaces does not interfere with the appointments. Furthermore, using both SRADS and appointments between two LPs can lead to further efficiencies.

$LP_1$ services $LP_2$'s SRADS read requests by simply returning the value of its logical clock. To process an appointment request. $LP$ must, at the very least. examine its clock. It follows that processing an SRADS read is no more costly to $LP_1$ than processing an appointment request. The clock value $LP_2$ observes in an SRADS read can always be used as an appointment, possibly replacing an older appointment time established in the conventional way. However, the read is cheaper than the appointment. Consequently, using both SRADS and appointments between two LPs is more efficient than using only appointments.

## 6. Conclusions

Distributed simulation on certain types of networks requires special synchronization protocols; the correctness and time accuracy of the simulation must be ensured as well as the deadlock freedom of the network. Such protocols have traditionally required the recipients of messages to passively await them. This approach easily leads to an undesirably high level of overhead for message writers. The SRADS protocol can avoid much of this overhead by employing active readers; SRADS however can lead to time inaccuracy. We have proposed the appointment, a protocol based on SRADS that ensures time accuracy. This protocol balances the network's need for synchronization information with the cost of providing that information. An empirical study of the appointment shows it to be more efficient than established approaches. The appointment and SRADS may be mixed within a network; an interface uses SRADS or the appointment according to its time accuracy sensitivity. We observe that this mixture can lead to further performance gains.

### *References*

(1) P.F. Reynolds, "A Shared Resource Algorithm for Distributed Simulation", *Proceedings of the Ninth Annual International Computer Architecture Conference*, Austin, Texas, 259–266, April 1982.

(2) J.K. Peacock, E. Manning, and J.W. Wong, "Synchronization of Distributed Simulation Using Broadcast Algorithms", *Computer Networks*, North Holland Publishing Co., 1980, 3–10.

(3) K.M. Chandy and J. Misra, "Asychronous Distributed Simulation via a Sequence of Parallel Computations", *Communications of the ACM*, 24, 11, 198–206, April 1981.

(4) D.R. O'Hallaron. "Analysis of a Model for Distributed Simulation", *M.S. Thesis. University of Virginia*, January, 1983.

(5) A.J. Croft, "Simulation of a Distributed Simulation", *M.C.S. Report, University of Virginia*, August, 1983.

(6) M.F. Theofanos. "Distributed Simulation of Queuing Networks", *M.S. Thesis, University of Virginia*, January, 1984.

(7) D.L. Davidson, "A Distributed Simulation Implimentation", *M.S. Thesis. University of Virginia*, January, 1984.

(8) D.L. Davidson, P.F. Reynolds, "Performance Analysis of a Distributed Simulation Algorithm based on Active Logical Processes", *Proceedings of the Winter Simulation Conference*, Arlington, VA, 267–270, December 1983.