Appeared in 9^{th} Workshop on Parallel and Distributed Simulation (PADS '95)

A Case Study in Simulating PCS Networks Using Time Warp

Christopher D. Carothers
and
Richard M. Fujimoto
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332

Yi-Bing Lin
Bellcore
Morristown, New Jersey 07962

Abstract

There has been rapid growth in the demand for mobile communications over the past few years. This has led to intensive research and development efforts for complex PCS (personal communication service) networks. Capacity planning and performance modeling is necessary to maintain a high quality of service to the mobile subscriber while minimizing cost to the PCS provider. The need for flexible analysis tools and the high computational requirements of large PCS network simulations make it an excellent candidate for parallel simulation.

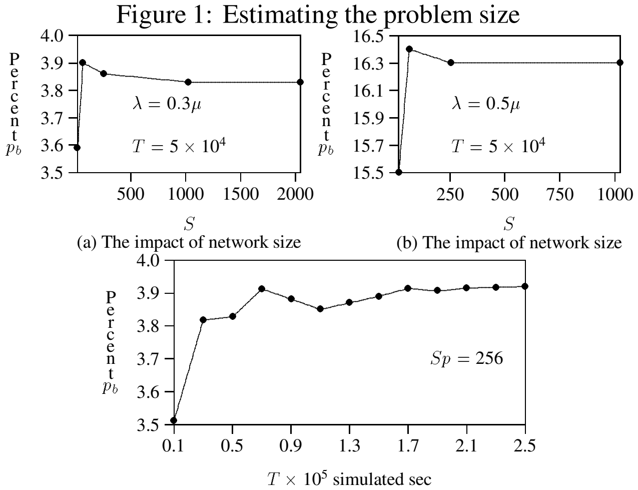
Here, we describe our experiences in developing two PCS simulation models on a general purpose distributed simulation platform based on the Time Warp mechanism. These models utilize two widely used approaches to simulating PCS networks: (i) the call-initiated *and (ii) the* portable-initiated *models. We discuss design decisions that were made in mapping these models to the Time Warp executive, and characterize the workloads resulting from these models in terms of factors such as communication locality and computation granularity. We then present performance measurements for their execution on a network of workstations. These measurements indicate that the call-initiated model generally outperforms the portable initiated model, but is not able to capture phenomenon such as the "busy line" effect. Moreover, these studies indicate that the high locality in large-scale PCS network simulations make them well-suited for execution on general purpose parallel and distributed simulation platforms.*

1 Introduction

A personal communication service (PCS) network [3] provides wireless communication services for nomadic users. The service area of a PCS network is populated with a set of geographically distributed transmitters/receivers called *radio ports*. A set of radio channels are assigned to each radio port, and the users in the *coverage area* (or *cell* for the radio port) can send and receive phone calls by using these radio channels. When a user moves from one cell to another during a phone call a *hand-off* is said to occur. In this case the PCS network attempts to allocate a radio channel in the new cell to allow the phone call connection to continue. If all channels in the new cell are busy, then the phone call is forced to terminate. It is important to engineer the system so that the likelihood of force termination is very low (e.g., less than 1%).

Traditionally, analytic models have been used to understand the performance characteristics of these types of networks. However, analytic models are often not sufficiently general or flexible because they must make simplifying assumptions about the system parameters to make the analysis tractable. For example, the blocking probability of a fixed channel assignment PCS network can be modeled analytically using the Erlang-B system [11, 20]. However, the Erlang-B system is only appropriate for predicting the blocking probability p_b when the

Figure 1: Estimating the problem size



(a) The impact of network size

(b) The impact of network size

(c) The impact of the termination time

number of portables is much larger than the number of available channels, resulting in a p_b that is much greater than 1%. Further, an analytic model for a fixed channel assignment PCS network (a network where the radio channels used in each cell are static and the number of channels per cell is constant) cannot be easily modified to study other aspects of a PCS network such as the effect of call hand-off [17] and dynamic channel assignment[1] [11, 20] on call blocking statistics. Now because discrete event simulation provides a general, flexible framework for evaluating system models, large high-capacity personal communication services (PCS) networks models are an excellent candidate for simulation.

In a PCS simulation, the network is usually represented by hexagonal or square cells. Ideally, PCS simulation experiments should model large networks containing thousands of cells. However, due to limited computing resources, most studies only examine small-scale networks containing fewer than 50 cells [11, 20], and output statistics of the boundary cells are usually discarded to avoid the boundary effect. Lin and Mak [12] showed that this approach may lead to biased output statistics, and suggest using a large network with a wrap-around topology to achieve reliable simulation results.

To illustrate the impact of network size, consider the following *portable initiated* simulation model using a fixed channel assignment scheme. A portable (or mobile phone) resides at a cell for a period of time, then moves to another cell. When a phone call arrives, a radio channel is used to connect the cor-

[1] Dynamic channel assignment strategies increase channel availability by "intelligently" reallocating channels to cells based on the traffic load within the cell, traffic mobility, and co-channel interference.

responding portable. If all channels are busy, then the call is blocked (dropped). The output measure is the blocking probability p_b, the probability that a phone call is dropped.

Figures 1 (a) and (b) illustrate the impact of network size. The experiments assume that the expected number of portables in a cell is N. A portable resides in a cell for a period of time t (in this paper, t has an exponential distribution with mean $\frac{1}{\eta}$, as proposed in [19]), then moves to one of the four neighboring cells with equal probability. The call arrivals to a portable form a Poisson process with the arrival rate λ. The call holding times have an exponential distribution with mean $\frac{1}{\mu}$. The simulation covers $T = 5 \times 10^4$ seconds of simulated time, the mean call holding time is $\frac{1}{\mu} = 180$ seconds, $N = 50$ and $\eta = 0.04\mu$.

Wrapping effect occurs when the statistical results produced from the simulation are in error because the cell boundaries "wrap" back on themselves. This wrapping of the cells boundaries allows a portable at an edge to suddenly move and reappear on the other side of the grid of cells. Because of this unnatural portable movement, error is introduced in the call statistics. This error can be reduced by using a larger network work of cells. The figures suggest that the network size $S > 256$ cells are required to avoid the wrapping effect.

Moreover, Figure 1 (c) suggests that $T > 5 \times 10^4$ (simulated) seconds are required to obtain the steady state blocking probability. For $S > 256$ and $T > 5 \times 10^4$ seconds, the number of events executed is on the order of 10^7 for a simple cell level simulation. For larger, more complex PCS simulations, such as Bellcore's Network Operations Plan (NOP) Architecture for PCS which includes integration with switching elements from the Plain Old Telephone Service (POTS) network, we estimate the number of events executed for a single run to be between 10^9 and 10^{10} depending on the detail of the simulation model. We have observed between 10^4 and 10^5 events per second in efficient discrete event simulation packages (commercial packages are often much slower). Accordingly, a simulation of this magnitude will require many hours of CPU time. Further, such large simulation programs may require more memory than is available on a single machine.

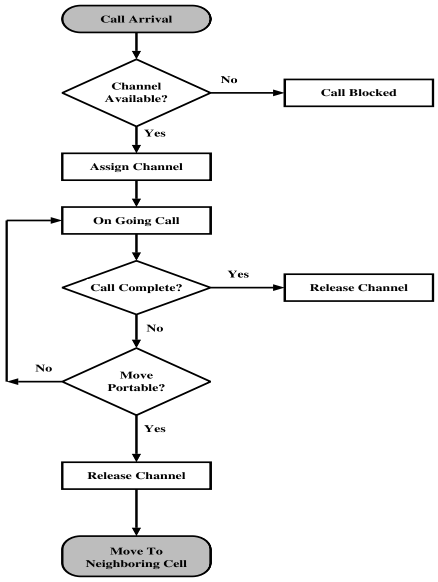
Because of the large computational requirements and the need for flexibility, PCS network models are a good candidate for a parallel simulation [8]. The impact of varying PCS simulation parameters on the Time Warp mechanism is investigated in [2]. This paper extends that work and examines our experiences in executing PCS simulation models on a distributed Time Warp system. In particular, we will discuss our approach to mapping the PCS models to the distributed simulator, characteristics of the workload model that impact the efficiency of the distributed simulator, and the performance of the system that was developed.

Most PCS models fall into two types of models: *call-initiated* [9, 17] or *portable-initiate* [19, 12]. In this study, we will consider both types of models. These models are described in section 2. Section 3 describes design decisions that were made to map the models to a Time Warp based distributed simulation environment. Section 4 characterizes the workload for both PCS models and discusses aspects of the workload that impact the distributed simulator. Section 5 describes the implementation details of our distributed Time Warp environment and presents performance data collected from executing the simulations on a collection of workstations. Section 6 concludes by reflecting on what was learned from this study about (i) the suitability of PCS simulations for parallel simulation, and (ii) what is important in the design and implementation of these models to enable efficient execution on a Time Warp system.

2 Description of the PCS Models

In this section we describe the *call-initiated* and *portable-initiated* models. The primary difference between these two models is that the *call-initiated* model only simulates the behavior of a portable during phone conversations. The *portable-initiated* model tracks the movement of portables even if they are not in conversations. The implication of these differences will be discussed in detail below.

Figure 2: *Call-Initiated Model*: flowchart for call processing within a single cell. A "Call Arrival" denotes a new call arriving at this cell or a call hand-off to this cell.

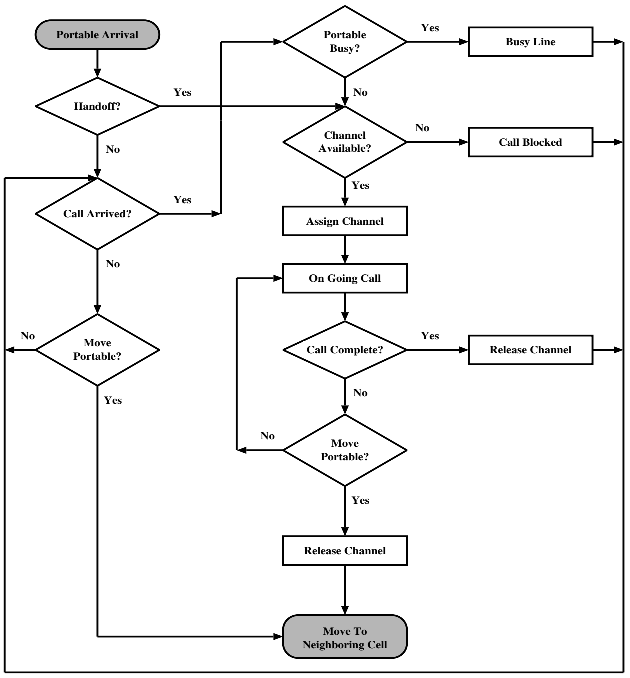


The *call-initiated* model is organized around two object types: *Cell* and *Call*. The Cell represents the area covered by a cellular receiver/transmitter that has a fixed number of channels allocated to it. The Call represents an active phone conversation. As depicted in figure 2, when a Call arrives at a Cell, channel availability is determined. If no channel is available then that Call is counted as being blocked. If a channel is available, it is assigned to that Call. When the Call completes, the occupied channel is released. If the portable to whom the Call is connected is leaving the Cell's area [2], then the allocated channel is released and the Call is handed-off to the destination Cell. The hand-off to the destination Cell is realized as a call arrival event at that neighboring Cell. If a channel at the neighboring Cell is not available, the Call is counted as a *hand-off block*, which is also known as a *forced termination*. If a channel is available, the Call reconnects and continues without interruption.

The *portable-initiated* model is also organized around two object types: *Cell* and *Portable*. As in the previous model, the Cell represents a cellular receiver/transmitter that has some fixed number of channels allocated to it. The Portable represents a mobile phone unit that resides within the Cell for a period of time and then moves to one of the four neighboring Cells. As shown in figure 3, when a new call arrives at a Cell, the Cell first determines the status of the destination Portable. If the destination Portable is busy with another call, this call is counted as a *busy line*. A *busy line* occurs when a Portable is currently connected in a phone call and another phone call arrives for that portable. The *call-initiated* model has no notion of a *busy line*, which is one of the fundamental differences between the two models and has an important effect on the call statistics. If the Portable is not busy, the Cell determines channel availability. If all channels are busy, the call is counted as a *call block*. If a channel is available, it is allocated for the destination Portable's use and the call is allowed to connect. While a call is in progress, the Portable tracks its location. When the Portable determines it is moving out of the current Cell's signal range, it drops the currently used channel, and requests a channel from the neighbor Cell into which it is moving. If all channels from the neighboring destination Cell are busy, this call counts as a

[2] We assume Portable movement is exponentially distributed.

Figure 3: *Portable-Initiated Model*: flowchart for call processing within a single cell. A' "Portable Arrival" denotes a portable entering a cell's area.



hand-off block. If a channel is available, the call reconnects and continues without interruption.

p_b is computed by the total number of blocked calls that are not *busy-lines* divided by the total number of call attempts. Because p_b should reflect the call blocking probability due to network congestion, *busy-lines* are subtracted from the total number of blocked calls prior to computing p_b. Table 1 shows that the blocking probability p_b, is significantly lower in this model than the blocking probability in the *call-initiated* model. The reason for this difference is *busy lines* are explicitly simulated in the *portable-initiated* model, which reduce the number of blocked calls in the simulation. By simulating *busy-lines*, calls that may have blocked due to insufficient network resources are being counted as *busy-lines*, which reduces the number of call blocks in computing p_b. This reduction in the number of blocked calls is called the *busy-line effect*. Accordingly, the *portable-initiated* model will have fewer blocked calls than the *call-initiated* model, because of *busy-lines*, for the same number of call attempts.

The difference in p_b for the two models will decrease as the number of portables per cell grows for some fixed number of channels per cell. As the number of portables per cell becomes larger, the number of call attempts grows, resulting in larger demands placed on network resources. As network demand rises, the p_b will rise and the impact of *busy-lines* on p_b will decrease. Accordingly, the *portable-initiated* model should be used when the number of portables per cell is not large because the busy line effect has a significant impact on p_b. For a large number portables per cell, the *call-initiated* model is more appropriate. It will be seen later that the *call-initiated* model is more efficient and can be simulated in less time than the *portable-initiated* model. The computational requirements for both models are discussed below.

3 Implementation

This section focuses on how we mapped the simulation model to the distributed simulator, emphasizing design and implementation decisions and rationales for these decisions that were made along the way.

3.1 Distributed Simulators

The distributed simulator consists of a collection of *logical processes* or LPs, each modeling a distinct component of the system being modeled. LPs communicate by exchanging timestamped event messages. Like most existing distributed simulation protocols, we assume different LPs may not share state variables that are modified during the simulation. The synchronization mechanism must ensure that each LP processes events in timestamp order in order to prevent events in the simulated future from affecting those in the past. The synchronization issue has been widely studied (e.g., see [4, 6, 13, 16]).

The Time Warp mechanism uses a detection-and-recovery protocol to synchronize the computation. For a more detailed discussion of the Time Warp mechanism we refer the reader to [4, 10].

3.2 Implementing the PCS Models

A natural way to map both PCS models to a general purpose, sequential simulator would be to realize each Call/Portable as a simulation process and Cells as state variables. In the *call-initiated* model, a call arrival at a cell could be simulated by creating a Call LP, which would persist until the call completes or is forced to terminate during a hand-off to another cell. In the *portable-initiated* model, a call arrival at a cell could be simulated by scheduling a "call arrival" event for a particular Portable LP. In this model, each Portable LP would persist for the duration of the simulation.

However, this approach poses a problem in Time Warp execution. In the *call-initiated* model, when a call is handed-off to another Cell, a channel at the destination Cell must be available. This implies that the Call LP must have access to a Cell's state information, in particular that Cell's channel availability. The Cell state information must therefore be shared among several Call LPs. But, such sharing state information is not allowed in Time Warp. A widely used solution to this problem is to implement each Cell as an LP. Accesses to the Cell LP's state information can be realized by exchanging timestamped messages between LPs, that carry the value(s) of requested state variable(s). However, in these PCS models, whenever a call arrives, the Portable/Call LP would need to exchange timestamped messages with the Cell LP to determine channel availability. The overhead associated with these messages becomes large, particularly when a Call or Portable has migrated to a Cell that is on a different processor. The cost is prohibitive in a distributed environment where communications are very expensive.

In order to avoid sharing state between LPs, we realized Cells as LPs but Calls/Portables as timestamped messages that travel among the Cell LPs. Mapping Cells to LPs is a standard technique used in other applications [18]. Mapping Call and Portables to timestamped messages is a reasonable approach from a modeling perspective when viewed from the model flowcharts (Figures 2 and 3). In both models when a call (portable) arrives, channel availability must be determined. Since the message denoting the call arrival is sent to the Cell LP in which the call will be processed, the channel availability information is accessible by the Call/Portable contained within the call arrival message. Moreover, in the *portable-initiated* model, Portable availability (is the Portable engaged in a phone call?) must also be determined by the Cell. Since the call arrival message carries the Portable's state information, Portable availability is known to the Cell. Using this mapping, a hand-off in both PCS models is realized as a message sent between two Cell LPs. The destination Cell LP views the hand-off as a call/portable arrival. Accordingly, during all phases of call processing for both PCS models, this mapping guarantees that the necessary state information is available without the additional overhead of exchanging timestamped messages.

A drawback with this approach is there are fewer LPs, and thus less parallelism in the simulator. In the present context, this is not a serious handicap because, as noted earlier, the required simulations will contain hundreds to thousands of Cells. Thus, there are a sufficient number of LPs for the computing platforms that were available for this study. Further, the amount of parallelism within a cell is limited, to a large extent, by dependencies arising from their accesses to the Cell LP's state information.

Table 1: Compare P_b for *call-initiated* and *portable-initiated* models across varied PCS parameters.

PCS Parameters	*call-initiated* p_b	*portable-initiated* p_b
$N = 25$, Mob = 1/(15 mins), Call = 30 mins	0.024%	0.011%
$N = 25$, Mob = 1/(15 mins), Call = 10 mins	10.92%	4.35%
$N = 25$, Mob = 1/(15 mins), Call = 6 mins	33.08%	17.52%
$N = 25$, Mob = 1/(45 mins), Call = 30 mins	0.021%	0.010%
$N = 25$, Mob = 1/(45 mins), Call = 10 mins	10.20%	3.79%
$N = 25$, Mob = 1/(45 mins), Call = 6 mins	32.46%	16.48%
$N = 25$, Mob = 1/(75 mins), Call = 30 mins	0.020%	0.007%
$N = 25$, Mob = 1/(75 mins), Call = 10 mins	10.10%	3.65%
$N = 25$, Mob = 1/(75 mins), Call = 6 mins	32.21%	15.99%

In both PCS models, the behavior of a Call/Portable is realized by 4 different types of events: (i) *NextCall*, which denotes a call arrival, (ii) *CompletionCall*, which denotes a completed call at a Cell (i.e. the call was not blocked), (iii) *MoveOut*, which denotes a call moving out of the current Cell, (iv) *MoveIn*, which denotes the arrival of a hand-off call at a Cell.

To initialize the *call-initiated* simulation, each Cell sends itself an initial event that invokes the *StartUp* method (event procedure). Upon processing the *StartUp* method, each Cell will initialize and schedule the arrival of the first Call to that Cell by scheduling an event into the future that will invoke the *NextCall* method. Each Call contains two independent timestamp fields: call completion, and move timestamps. The call completion timestamp represents the time at which the phone call will complete. The move timestamp represents the time at which the Call will leave the current Cell and reside at one of the neighboring Cells. The minimum of these two timestamp fields determines the event type and Cell destination of the Call. Call arrivals are generated by scheduling the next Call arrival at the end of processing the *NextCall* method. Accordingly, because each Cell LP determines its own Call arrivals and sends messages to itself, this simulation is *self-initiating*. This impacts the performance of the distributed simulation, as will be discussed later. For a detailed analysis of the performance bounds for *self-initiating* simulations, we refer the reader to [15].

The *portable-initiated* simulation is similar to the *call-initiated* Model. During initialization, each Cell sends itself an initial event that invokes the *StartUp* method. Upon processing the *StartUp* method, each Cell will initialize and schedule some fixed number of messages, each containing a Portable. Each Portable contains three independent timestamp fields: call completion, next call and move timestamps. In the *call-initiated* model, Calls only contained two independent timestamps: call completion and move timestamps. The call completion and move timestamps have the same meaning in both models. However, the additional next call timestamp contained in the *portable-initiated* simulation represents the time when the next call will arrive. The primary purpose of the next call timestamp is to determine when *busy lines* occur. The minimum of these three timestamp fields determines the type of event that will be scheduled by a Cell. Portable arrivals are generated by determining when a Portable will receive its next call arrival based on the NextCall timestamp. Accordingly, like the *call-initiated* model, the *portable-initiated* model is also *self-initiating*.

In both models, Call/Portable interarrival times, Call/Portable completion times and Call/Portable move times are exponentially distributed, though one could trivially substitute other distributions into the simulation program.

4 Workload Characterization

In this section we characterize the workload for each PCS model. First we quantify the workload in terms of the PCS model application parameters, and then discuss characteristics that are relevant to the performance of the distributed simulator.

4.1 Model Parameters

Both PCS models have the following application parameters: (i) Call/Portable mobility, (ii) call interarrival time, (iii) number of Cells, (iv) number of Calls/Portables. Each of these parameters is discussed below.

Mobility of Calls/Portables determines how frequently Calls/Portables move to a different Cell. This, in turn, determines how frequently LPs communicate. Recall from the previous discussion that the only time Cell LPs communicate is in the handoff of a Portable/Call. As LP's communicate more frequently, the number of messages sent between processors increases, which increases distributed simulator overheads, leading to slower execution times. Section 5 will present performance data that supports this hypothesis.

The call interarrival time determines the amount of work available to the simulator over a given period of simulated time. For modest size PCS networks, the call interarrival time has a significant impact on the "rate" the simulation progresses through simulated time, which will determine how likely a simulation will roll back for a fixed amount of lookahead. The faster the rate of progress, the more likely the simulation will roll back. Lookahead is defined as the amount of simulated time an LP can "see" into the future and will be discussed in more detail later. For large PCS networks, the workload is sufficient that a change in call interarrival times will not impact the simulation's rate of progress to effect the likelihood that rollbacks occur. Section 5 will present performance data that supports this hypothesis.

The number of Cells determines the number of LP's in the Time Warp simulation. 1024 LPs were used for all experiments presented in this study. Because of the large number of LPs and self-initiating nature of the simulation implementation, the amount of available parallelism should be very high. Also, the number of Cell's determines the amount of memory needed for state saving.

Next, the number of Portables in the *portable-initiated* model determines the number of pending events. For experiments presented in this study, N, the number of Portables per Cell, is varied among $N = 25, 50, 75$. Accordingly, the total number of pending events in the simulation ranges from $25,000$ to $75,000$. By contrast, the number of Calls in the *call-initiated* model does not determine the number of pending event. To model N Portables per Cell in the *call-initiated* model, the call interarrival time is the call interarrival time of the *portable-initiated* model divided by the number of Portables per Cell. Also, in the *call-initiated* model, a call only persists as long as the conversation lasts or until it is forced to terminate. Accordingly, the number of pending events is bounded by the number of channels in the simulation. For all experiments presented in this study, the number of channels per Cell was fixed at 10, making the number of pending events in the simulation on the order of $10,000$ for any number of Portables.

Additionally, the number of Calls/Portables per Cell for both PCS models impacts the rate at which simulated time advances. Increasing the number of Portables in the *portable-initiated* simulation results in increasing the amount of work an LP must perform. Similarly, increasing the number of Calls in the *call-initiated* simulation increases the rate at which calls arrive with respect to simulated time which increases the amount of work an LP must perform.

4.2 Workload Properties

In general, the following properties can significantly impact the performance of Time Warp based, distributed simulators: (i) lookahead of LPs, (ii) event granularity, (iii) state saving overheads, (iv) message data size, (v) locality of communications, and (vi) homogeneity of the model being simulated.

As previously stated, lookahead determines how "far" into the simulated future an LP can predict future events. In [5] it was shown that lookahead can significantly affect the performance of Time Warp simulators with small message densities. However, since both PCS models contain very large message densities, the impact of lookahead on system performance may not be as large. Because timestamps for determining call interarrivals, call completions and mobility are exponentially distributed, both model implementations will have a lookahead of 0 in the worst case. Lookahead can be improved by pre-computing portions of the computation for future events [14]. For the PCS models discussed in this study, it is possible to pre-compute call completion times, which should result in an increase in the average lookahead of the simulation. However, this improvement reduces the extensibility of the simulation implementation to incorporate new features added to the simulation model. For example, the Bellcore's Generic Criteria for 0.1 Wireless Access Communications System (WACS) allows an active call to be interrupted for emergency purposes, such as the processing of a 911 call when no channel is available. This feature could not be easily added to the "improved" simulation implementation without advanced knowledge of when a call would be interrupted. For this reason, techniques for improving lookahead were not exploited in this implementation. Thus, the performance that was obtained is conservative in this regard.

Event granularity is the average amount of real time to compute an event. If the event granularity is small then incorrect computation tends to propagate more rapidly relative to the speed of the correction computation (anti-messages). Table 2 shows the event granularity, excluding message send times, for each event type for the different PCS models. These timings where obtained by running the simulation on a single Sun Sparc-2 workstation. The timing information was captured by the Unix *gettimeofday* system call, which has a resolution of 30 microseconds on this computing platform. A significant portion of this granularity is overhead introduced by C++. We are currently examing ways to reduce this overhead.

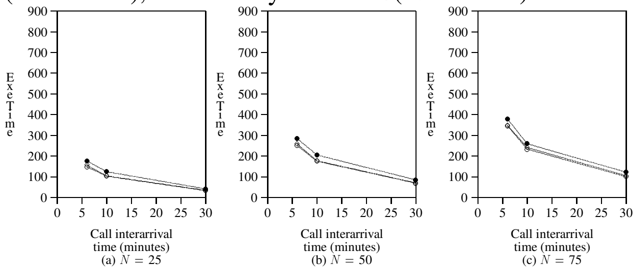
Because the Cell LPs of both PCS models contain less than 256 bytes of state, the state saving overhead is modest.

Scheduling an event requires the message data to be copied into a new message buffer. Accordingly, the amount of data contained in a message affects the overhead of scheduling events as well as the amount of memory need to efficiently execute the simulation. Portables and Calls contain less than 256 bytes of data, so the overhead in scheduling events is low. However, the number of pending events in the system may be as large as $75,000$, necessitating a minimum of almost 20 MB memory for LP state and event data.

The locality of each PCS simulation is shown in Table 3. We see that most event types in the *call-initiated* model schedule events to *self* while the *portable-initiated* model can schedule an event to either *self* or a neighbor LP (north, south, east, and west). This difference is due to a Portable persisting for the life of the simulation in the *portable-initiate* model, even during periods of inactivity (i.e. not connected to an ongoing call), while a Call in the *call-initiated* model is terminated if the Call completes or is blocked.

The homogeneity of a model determines how easily the simulation objects (LP's, number of pending events) are mapped onto the processors and how evenly distributed the workload is across the processors. For both PCS models, since all the Cells are identical (same mean and distribution for call interarrivals call completions, mobility times and same number of Portables per Cell), the workload will be well balanced across all processors, thus eliminating the need for dynamic load management mechanisms.

Figure 4: *Call-Initiated*: parallel execution time (in seconds) versus number of portables per cell (N), mobility and call interarrival times. Call holding times are exponentially distributed with a 3 minute mean. Experiments performed using 8 processors. • : Mobility mean = 1/(15 minutes), ⋄ : Mobility mean = 1/(45 minutes), ∘ : Mobility mean = 1/(75 minutes).



5 Performance Results

In this section we discuss our distributed implementation of the Time Warp mechanism and then present performance data for the two PCS simulation models.

5.1 Implementation of Distributed Simulator

A version of Time Warp has been developed that executes on a collection of DEC 5000 workstations, Sun Sparc workstations, or a mixture of these machines. All performance results presented here were performed on DEC machines interconnected by an Ethernet. The Time Warp system is written in C++. A principal objective of this implementation is to enable efficient simulation of thousands of "light weight" LPs (i.e., processes that contain a small amount of state and perform little computation in each event) in an object-oriented environment on networked, heterogeneous workstations. Here we will describe some of the user and kernel level features of the Time Warp system.

LPs are implemented as C++ objects, and are referred to as "entities." Each LP (entity) consists of a state vector that stores the LP's private data, and a set of methods that define the allowable operations that can be performed on that data. Each method corresponds to a type of event. For instance, in the PCS simulations, an entity for a cell will include the methods described earlier in section 3.2 (*NextCall, CompletionCall, MoveOut, MoveIn*).

It is anticipated that most simulations will contain far more entities than processors, so each processor will typically contain hundreds or thousands of entities (LPs). A priority queue data structure called the calendar queue [1] is used in each processor to select the next entity to execute. The processor's scheduler always selects the entity containing the smallest timestamped event as the next one for execution. Each entity includes a linear list to hold the unprocessed events (method invocations) scheduled for that entity.

To avoid unnecessary system overheads from `malloc` system calls and memory fragmentation, the Time Warp kernel allocates a single contiguous block of memory from which events and other internal data structures are allocated.

Communications between processors is implemented using PVM 3.2 (Parallel Virtual Machine), a message passing system for heterogeneous collections of networked computers [7].

In addition to PVM's default routing of messages, PVM will route messages directly between application processes by setting up a TCP/IP connection between each enrolled application, thus bypassing the PVM daemons at both the sending and receiving host. In this mode, the sending process is not blocking, making PVM's direct routing mode an asynchronous communications protocol. When routing messages in direct mode, the total time to deliver a message is in the range of 1.4 to 2.0 milliseconds. When executing our distributed Time Warp kernel, PVM is configured to directly route messages.

Table 2: Comparison of event types for *call-initiated* model and *portable-initiated* model based on event granularity

Event Type	*call-initiated* event grain	*portable-initiated* event grain
NextCall	258 usec	125 usec
CompletionCall	38 usec	93 usec
MoveCallIn	196 usec	107 usec
MoveCallOut	195 usec	324 usec

Table 3: Comparison of event types for *call-initiated* model and *portable-initiated* model based on number of messages sent and locality of destination.

Event Type	*call-init* # msgs	*portable-init* # msgs	*call-init* dest	*portable-init* dest
NextCall	2	1	self	self or near neighbor
CompletionCall	0	1	NA	self or near neighbor
MoveCallIn	1	1	self	self or near neighbor
MoveCallOut	1	1	near neighbor	near neighbor

Figure 5: *Portable-Initiated*: parallel execution time (in seconds) versus number of portables per cell (N), mobility and call interarrival times. Call holding times are exponentially distributed with a 3 minute mean. Experiments performed using 8 processors. • : Mobility mean = 1/(15 minutes), ⋄ : Mobility mean = 1/(45 minutes), ∘ : Mobility mean = 1/(75 minutes).

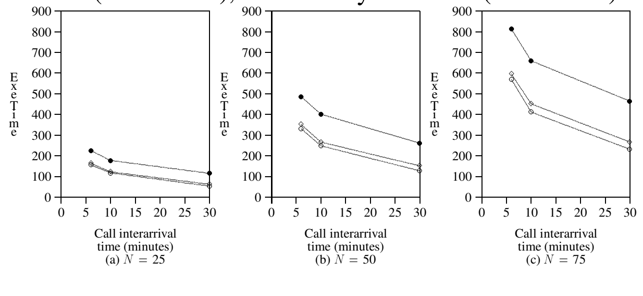
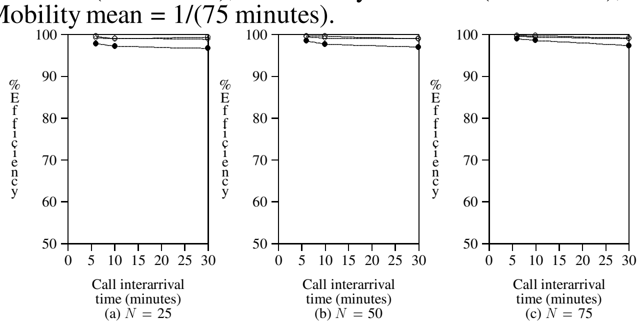


Figure 6: *Call-Initiated*: efficiency of simulation versus number of portables per cell (N), mobility and call interarrival times. Call holding times are exponentially distributed with a 3 minute mean. Experiments performed using 8 processors. • : Mobility mean = 1/(15 minutes), ⋄ : Mobility mean = 1/(45 minutes), ∘ : Mobility mean = 1/(75 minutes).



5.2 Performance Data

To determine the effect of the PCS workload characteristics on Time Warp performance, we measured the execution times, efficiency of the simulation [3], average rollback distance and percentage of remote communications for both PCS models. For each metric, the number of portables per cell, mobility, and call interarrival times were varied. The completion time for a phone call was exponentially distributed with mean 3.0. The number of channels per cell was fixed at 10. All experiments were conducted on 8 DEC 5000 workstations, with each workstation containing 128 cells, for a total of 1024 cells. All experiments ran for 2.5×10^4 simulated seconds. Each data point represents the average of 6 runs.

In Figures 4 and 5, the execution time is shown for each of the PCS parameter values (number of portables, mobility, and call interarrival times). In every case the *call-initiated* simulation outperformed the corresponding *portable-initiated* simulation. This can be attributed to the *call-initiated* model's sending fewer remote messages, which is due to not simulating the inactive periods of portables. Remote communications is discussed later in this section.

Combined with fewer remote messages, there were fewer events processed in the *call-initiated* simulation for all cases except where the mean call interarrival time was low (6 minutes). The reason more events are being processed in this case by the *call-initiated* simulation is because calls that would be counted as *busy lines* in the *portable-initiated* simulation, due to the high arrival rate of calls, are allowed to connect in *call-initiated* simulation. However, when the call interarrival time is longer, the movement of call-less portables in the *portable-initiated* simulation causes the processing of more events than in the *call-initiated* simulation.

The large size of the simulation models precluded reliable measurement of sequential execution times. Memory management overheads artificially inflated the execution time, causing super-linear speedups to be obtained. However, based on execution of smaller sized models, we estimate speedups range between 4 and 8 for the distributed simulator.

Despite significant differences in execution times, the efficiencies for both PCS simulations, as shown by Figures 6 and 7, are over 95% for all PCS parameters (number of portables per cell, call interarrival times, and mobility). These high efficiencies are attributed to the size of the simulation workload relative to the number of processors in both PCS implementations. Because interprocessor communications are relatively infrequent, the frequency of rollbacks is diminished.

Figures 8 and 9 show the average rollback distances for the *call-initiated* and *portable-initiated* simulations. We see that both simulations experienced long rollbacks. The *call-initiated* simulation had the shortest of the two simulations with rollbacks distances ranging from 2.6 to 6.5. In the *portable-initiated* simulation, rollback distances ranged from 3.5 to 7.8. These results are attributed to the *self-initiating* nature of both simulations. Because, LPs schedule many events to themselves, neighboring LPs become "unsynchronized" with respect to simulated time,

[3]Efficiency is defined as the percentage of the simulation computation that is committed.

Figure 7: *Portable-Initiated*: efficiency of simulation versus number of portables per cell (N), mobility and call interarrival times. Call holding times are exponentially distributed with a 3 minute mean. Experiments performed using 8 processors. • : Mobility mean = 1/(15 minutes), ⋄ : Mobility mean = 1/(45 minutes), ∘ : Mobility mean = 1/(75 minutes).

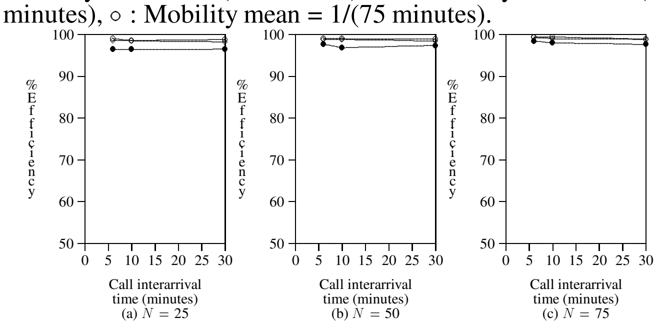


Figure 8: *Call-Initiated*: average rollback distance measured in events per rollback versus number of portables per cell (N), mobility and call interarrival times. Call holding times are exponentially distributed with a 3 minute mean. Experiments performed using 8 processors. • : Mobility mean = 1/(15 minutes), ⋄ : Mobility mean = 1/(45 minutes), ∘ : Mobility mean = 1/(75 minutes).

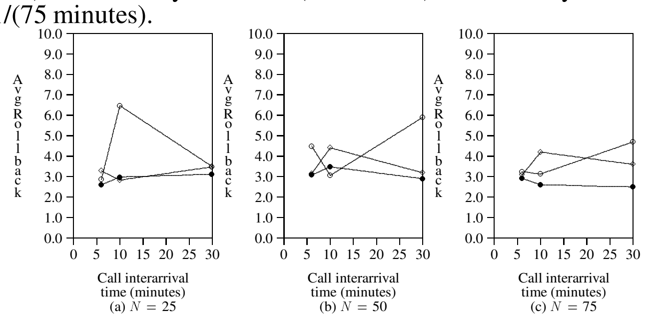


Figure 9: *Portable-Initiated*: average rollback distance measured in events per rollback versus number of portables per cell (N), mobility and call interarrival times. Call holding times are exponentially distributed with a 3 minute mean. Experiments performed using 8 processors. • : Mobility mean = 1/(15 minutes), ⋄ : Mobility mean = 1/(45 minutes), ∘ : Mobility mean = 1/(75 minutes).

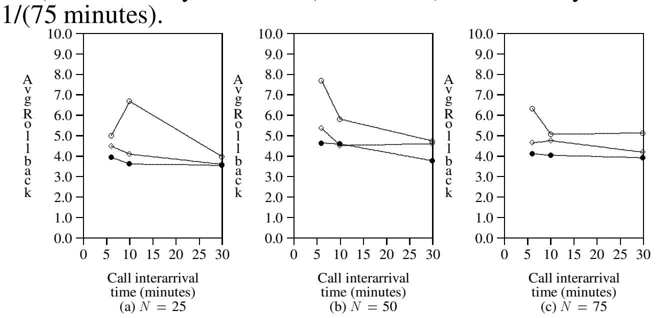
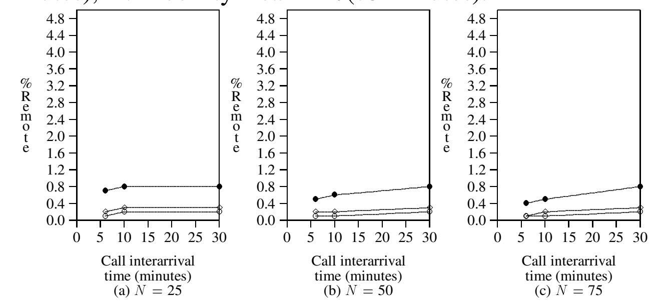


Figure 10: *Call-Initiated*: % remote communications versus number of portables per cell (N), mobility and call interarrival times. Call holding times are exponentially distributed with a 3 minute mean. Experiments performed using 8 processors. • : Mobility mean = 1/(15 minutes), ⋄ : Mobility mean = 1/(45 minutes), ∘ : Mobility mean = 1/(75 minutes).



leading to longer rollbacks when LPs exchange messages. However, because the efficiency of both simulations is very high, the number of rollbacks is extremely low.

The "spikes" in Figures 8(a),(b),(c) and 9(a) are attributed to a shift in rollback behavior. When increasing the mean call interarrival time from 6 minutes to 10 minutes, the number of secondary rollbacks increases because the workload has decreased, allowing the LPs to become more "unsynchronized". We believe these rollbacks tend to be very long and perturb the average. However, when increasing the call interarrival time mean from 10 minutes to 30 minutes, we believe the frequency of secondary rollbacks increase again, but become shorter as a result of being more frequent, which reduces the average rollback distance.

In Figures 10 and 11, the percentage of committed events that were sent to LPs on different processors is shown. We see that the *call-initiated* simulation has less than 1% remote communications, while the *portable-initiated* model ranges from 0.6% to 5.0% remote communications, depending on the mobility. The cause of these differences in remote communications lies with the "life time" of Calls in the *call-initiated* simulation and Portables in the *portable-initiated* simulation. Recall, in the *call-initiated* implementation, that when a Call completes or is blocked that Call ceases to exist. However, in the *portable-initiated* simulation, Portables persist for the duration of the simulation and move between Cells even when not processing a phone call, resulting in more remote messages sent between LPs. Because a higher percentage of messages are sent between LPs residing on different workstations combined with the high cost of remote communications, the communications overhead in the *portable-initiated* simulation are much larger than *call-initiated* simulation, which underscores the execution time results presented in Figures 4 and 5.

6 Conclusions

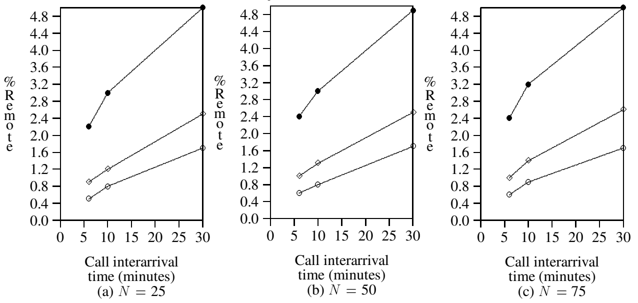
In this paper we characterize workloads for *call-initiated* and *portable-initiated* PCS models in terms of suitability for Time Warp execution. Experiments were completed to determine how the workload characteristics affect Time Warp execution on a collection of networked workstations. From these experiments, we conclude:

- An important aspect of designing and implementing these models (and any simulation model in general) is developing a suitable mapping to Time Warp's execution paradigm, such as eliminating the need for sharing state among LPs and placing model objects to maximize the locality of communications (i.e., implementing simulations to be *self-initiating*) as was done in the implementation of the PCS models.

- It was shown that, despite a poor lookahead, both PCS simulations exhibited over 95% efficiencies for all PCS parameters. Accordingly, lookahead does not seem to play as significant a role in determining performance for these large workload simulations as it does for smaller models.

- Efficient execution of small granularity, Time Warp simulations is possible on distributed environments. This suggests that the aggregate workload of the simulation model and not the event granularity should be used when determining suitability for Time Warp execution.

Figure 11: *Portable-Initiated*: % remote communications versus number of portables per cell (N), mobility and call interarrival times. Call holding times are exponentially distributed with a 3 minute mean. Experiments performed using 8 processors. • : Mobility mean = 1/(15 minutes), ⋄ : Mobility mean = 1/(45 minutes), ∘ : Mobility mean = 1/(75 minutes).



- Under modest remote communications loads, large message delivery times for asynchronous communications protocols do not appear to have a significant impact on Time Warp efficiencies for large workloads.

References

[1] R. Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, Oct. 1988.

[2] C. D. Carothers, R. M. Fujimoto, Y-B. Lin, and P. England. Distributed Simulation of Large-Scale PCS Networks. In 2^{nd} *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 2–6. IEEE Computer Society Press, Feburary 1994.

[3] Cox, D. C. Personal Communications – A Viewpoint. *IEEE Communications Magazine*, 128(11), 1990.

[4] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[5] R. M. Fujimoto. Performance of Time Warp under synthetic workloads. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 23–28, 1990.

[6] R. M. Fujimoto and D. M. Nicol. State of the art in parallel simulation. In *1992 Winter Simulation Conference Proceedings*, pages 122–127, December 1992.

[7] A. Geist, A. Beguelin, J. Dongarra, W. Jian, R. Manchek, and V. Sunderam. Pvm 3 user's guide and reference manual. Technical Report TM-12187, Oak Ridge National Laboratory, May 1993.

[8] A. G. Greenburg, B. D. Lubachevsky, and D. M. Nicol. Efficient Massively Parallel Simulation of Dynamic Channel Assignment Schemes for Wireless Communications. In 8^{th} *Workshop on Parallel and Distributed Simulation*, volume 24. SCS, July 1994.

[9] J. C-I Chuang. Performance issues and algorithms for dynamic channel assignment. *IEEE J. Select. Areas Commun.*, 1993.

[10] D. R. Jefferson. Virtual time. *ACM TOPLAS*, 7(3):404–425, July 1985.

[11] Kuek, S.S., and Wong, W.C. Ordered Dynamic Channel Assignment Scheme with Reassignment in Highway Microcells. *IEEE Trans. Veh. Technol.*, 41(3):271–277, 1992.

[12] Lin, Y.-B., and Mak, V.K. On Simulating a Large-Scale Personal Communications Services Network. To appear in ACM TOMACS, 1993.

[13] J. Misra. Distributed-discrete event simulation. *ACM Computing Surveys*, 18(1):39–65, March 1986.

[14] D. M. Nicol. Parallel discrete-event simulation of FCFS stochastic queueing networks. *SIGPLAN Notices*, 23(9):124–137, September 1988.

[15] David M. Nicol. Performance bounds on parallel self-initiating discrete-event simulations. *ACM TOMACS*, 1(1):24–50, January 1991.

[16] R. Richter and J. C. Walrand. Distributed simulation of discrete event systems. *Proceedings of the IEEE*, 77(1):99–113, January 1989.

[17] Tekinary, S., and Jabbari, B. A Measurement Based Prioritization Scheme for Handovers in Cellular and Microcellular Networks. *IEEE J. Sel. Areas Comm.*, October 1992.

[18] F. Wieland, L. Hawley, A. Feinberg, M. DiLorento, L. Blume, P. Reiher, B. Beckman, P. Hontalas, S. Bellenot, and D. R. Jefferson. Distributed combat simulation and Time Warp: The model and its performance. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 14–20. SCS Simulation Series, March 1989.

[19] Wong, W. C. Packet Reservation Multiple Access in a Metropolitan Microcellular Radio Environment. *IEEE J. Sel. Areas Comm.*, 11(6):918–925, 1993.

[20] Zhang, M., and Yum, T.-S. Comparisons of Channel-Assignment Strategies in Cellular Mobile Telephone Systems. *IEEE Trans. Veh. Technol.*, 38(4):211–215, 1989.