Time-parallel simulation of wireless ad hoc networks with compressed history

Guoqiang Wang, Ladislau Bölöni, Damla Turgut and Dan C. Marinescu
School of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL 32816-2362

May 11, 2008

Abstract

Time-parallel simulation (TPS) is a simulation technique which partitions the timespan of the simulation into independently executed simulation segments. Unless the simulated process is regenerative, the output of TPS is only an approximation of the corresponding serial simulation. In previous work, we have adapted TPS to the simulation of wireless ad hoc networks. By prefixing the measured simulation segment with a warmup interval which can be dynamically extended, we were able to achieve arbitrary accuracy. In general, higher accuracy requires a longer warmup interval, which decreases the speedup.

In this paper we introduce *compressed history*, a technique which improves the performance of TPS for a class of processes which require long warmup intervals to achieve satisfactory accuracy. Compressed history replaces part of the warmup interval, and speeds up the simulation by retaining only those past events that affect significantly the state at the beginning of the measured interval. We present compressed history in general terms, and provide a detailed description of its implementation, tuning and performance results for the concrete example of the DSDV proactive ad hoc routing protocol.

1 Introduction

Discrete event simulation allows us to investigate the functionality as well as the performance of computer and communication systems before such systems are actually built. The simulation program processes individual events that trigger changes of the state. This allows us to observe the state of the model throughout the simulated time interval and to carry out "measurements" for significant performance data.

For instance, by simulating a wireless network, we can estimate packet loss ratio, throughput, jitter and other network performance metrics. Such simulations are valuable tools in the provisioning of new wireless network deployments, as performance problems can be detected ahead of time. Unfortunately, the simulation of large deployments of wireless networks is computationally expensive. The simulation can be an order of magnitude slower than the simulated process, making the testing of various alternative designs difficult. Ad hoc wireless networks, where nodes act as routers are especially expensive to simulate.

A natural solution would be to use parallelism to improve the simulation time. Unfortunately, the simulation of wireless networks is not easy to parallelize. Many parallel simulation approaches partition the *space* of the simulation scenario. These approaches provide the best speedup if the simulation process exhibits strong spatial locality - that is, the effects of an event stay mostly in its vicinity. Unfortunately, every wireless transmission affects the entire transmission range of the node, thus the spatial locality criteria is not satisfied. In most cases, however, there is relatively little *temporal* dependency among the wireless transmissions. Thus, time-parallel simulation (TPS) appears to be a good choice for wireless network simulation.

In the simplest form of time-parallel simulation, the timespan of the simulation is partitioned into smaller adjacent simulation segments (Figure 1-a). The initial state of each segment is *guessed* and each segment is simulated on a separate processor. The accuracy of the time-parallel simulation depends on our ability to estimate the state of the system at the beginning of the simulation segment. With a perfect estimation of the initial state, the speedup is limited only by the number of available processors. One case when the initial state can be perfectly estimated is when the state is empty, which happens if the stochastic model of the system is *regenerative* and the time partitioning happens at the regeneration points. This, however, is a situation rarely encountered in wireless ad hoc networks.

In our previous work [1–4], we applied TPS to the simulation of wireless ad hoc networks and proposed the combination of a priori initial state approximation and a simulation warmup interval. The warmup interval technique works as follows. For each simulation segment we consider two intervals: the warmup interval and measurement interval (see Figure 1-b). During the warmup, we perform the simulation but do not carry out any measurements; during the second phase we perform measured simulation, where the initial state of the simulation is approximated by the final state of the warmup interval. This initial state takes into account perturbing events occurring during the warmup phase, which would have been otherwise ignored.

A warmup interval approximates the initial state because most perturbations caused by network events will eventually diminish or extinguish. In [1] we presented a layer-by-layer analysis of the impact of perturbations on the functioning of the wireless network. We found that the perturbations with the most significant and longest lasting impact are those which affect the routing tables.

Naturally, the longer the warmup interval, the better the approximation. In our methodology, the accuracy of the simulation is gradually increased by extending the warmup interval through a series of iterations. The simulation will converge to the accurate simulation in finite time (which might not, however, be a speedup compared to the serial simulation). We prefer convergence curves which show a good approximation early, rather than maintaining a large error throughout the process of warmup extension and dropping to zero at the last moment. The simulation process is stopped when the desired accuracy is reached; in practice, we found that a 95% accuracy is sufficient for most applications. The typical speedups obtained were about 10-16 times for ad hoc networks with reactive routing protocols such as AODV [5] and 4-5 times for proactive routing protocols such as DSDV [6]. Note that the proactive protocols, which carry more state information than the reactive ones, need longer warmup intervals for equivalent accuracy, thus limiting the achievable speedup.

In this paper we improve the speedup of time-parallel simulation, by replacing the warmup interval completely or partially with a *compressed history* of the simulation. The warmup interval represents the complete simulation of a time period (a history) before the measured part of the simulation. From this simulation, we only retain the final state, which will serve as the initial state for the measurement interval. The idea behind compressed history is that we can replace the warmup interval with a simplified, computationally cheaper simulation scenario, which might not yield the same simulation trace, but will leave the network in the same (or very similar) final state. This technique is illustrated in Figure 1-c.

The technique of compressed history is quite general and can be applied for time-parallel simulation in any domain. On the other hand, the creation of a compressed history that provides a good approximation of the final state but is computationally cheaper, is necessarily application dependent. We need to apply domain specific knowledge about what type of events have an impact on the final state of the simulation and which are the events which can be removed or substituted with simpler events. It is also critical to focus on removing the events whose simulation is computationally the most expensive.

To anchor our presentation into a concrete example, in this paper we will concentrate on applying compressed history to the simulation of the DSDV proactive routing protocol. Our previous work demonstrated that DSDV can be simulated using TPS, but the maximum speedup is limited to 4-5 times, smaller than, for instance the 10-16 times speedup in the simulation of AODV. In this respect, DSDV which carries relatively strong dependency on the history encoded in its routing tables, is a good candidate for history compression. If we would apply compressed history to protocols which are much less dependent on history, such as AODV, the technique would still work, but the speedup would be smaller – as there is less opportunity for compression.

The reminder of this paper is organized as follows. Related work is presented in Section 2. We provide a formal model of the compressed history technique in Section 3, while our implementation is detailed in Section 4. A simulation study is described in Section 5. We discuss the generality of the approach in Section 6 and conclude in Section 7.

2 Related work

Improving simulation speed through parallelism has been in the attention of researchers since the mid 1970s. A particular challenge is to implement the simulations on distributed environments, which assume loosely connected computers without shared memory. Most of the work concentrated on parallel discrete event simulation (PDES) [7–11] which assumes that the phenomena to be simulated evolves through discrete events. The simulation of the events affects the state variables of the system, and also creates events in the future, which can be seen as the *effect*

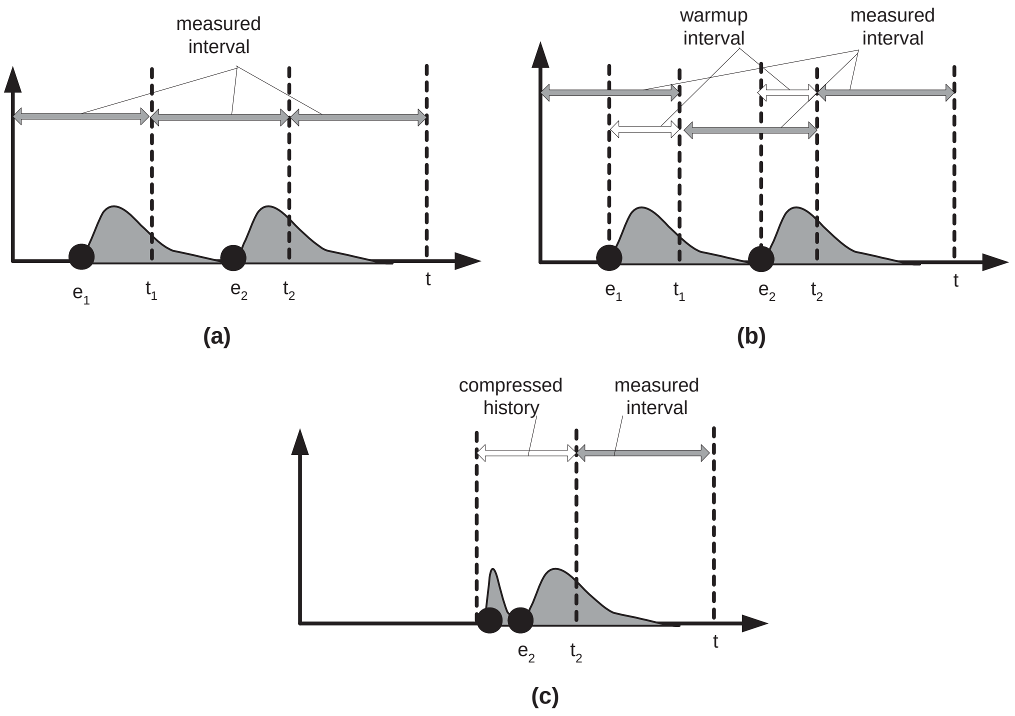


Figure 1: The scope of perturbations and techniques to improve the initial state estimation in time-parallel simulation. (a) A perturbation propagates beyond the boundary of the interval it originated from, and no attempt is made to estimate the effect of it upon the initial state of successor intervals. (b) Warmup interval. (c) Warmup with compressed history, selected past events are used to estimate the initial state.

of the simulated events. The simulation of the events is distributed across multiple logical processors. To guarantee the correctness of the simulation, each logical processor must simulate the events in the correct order - this is the *synchronization problem*. There are two basic approaches for solving the synchronization problem. Conservative synchronization waits with the simulation of each event until it is safe to proceed; the earliest implementation being the Chandy-Misra-Bryant algorithm [12, 13]. The appropriate choice of the partitioning of the simulation domain across multiple processing units has a significant impact on the performance. A static partitioning approach which takes into account explicitly the cost of the Chandy-Misra null messages is [14]. Further improvements can be obtained by dynamic load balancing of the simulation [15].

Optimistic synchronization allows the event to proceed, but if local causality violations occur, they are detected at runtime and recovered using a rollback mechanism. An example is Jefferson's Time Warp algorithm [16].

For both approaches, a factor of critical importance is the *lookahead* [17], the minimum delay between an event and its effects. In general, the larger the lookahead, the more efficient the simulation. In the case of simulation of physical systems, sometimes a large lookahead can be achieved by space partitioning. If an event is an agent boarding an airplane at a certain location, the effect event, the arrival to the remote location, will happen after a delay of several hours. However, if instead of flying an airplane, the agent calls by phone, the delay between cause (dialing a number) and effect (the remote phone ringing) is much shorter. The major challenge of PDES for the wireless networking domain is that if two nodes are in each other's transmission range, the delay between the sending and receiving events is very short, which makes both conservative and optimistic synchronization inefficient. Systems using PDES in the wireless networking domain exploit domain specific knowledge to improve synchronization efficiency.

The SWiMNet simulator [18, 19] simulates Personal Communications Service (PCS) networks with fixed channel allocation. It allows a rich model including fine grained mobility, variable cell process, and arbitrary coverage area. It is based on a combination of optimistic and conservative paradigms which exploits model independence in the PCS networks.

Another system simulating PCS networks using the optimistic paradigm of asynchronous parallel distributed event simulation is described in [11]. The system models the PCS coverage area using fixed hexagonal cells.

The PDNS [20] project implements a parallel version of the NS-2 simulator [21]. However, the applicability of PDNS is limited to *wired* networks as the traffic simulated at different spatial partitions cannot affect each other.

Time-parallel simulation (TPS) also considers phenomena which evolves through discrete events. However, instead of exploiting the delay between the cause event and its effects, as PDES does, it relies on the (quasi-) independence of events happening at remote time points. Naturally, not every type of phenomena can be simulated using this model; TPS is based on the assumption that every event causes a chain of effects (the *perturbation* in the system) of finite length.

In time-parallel simulation, the simulated time interval is partitioned into smaller adjacent intervals. Each interval is assigned to a processor together with a *guessed* initial state [22–27] and the simulation terminates when the final state of each interval matches the initial state of its successor. State matching is one of the key problems of time-parallel simulation. A simulation is defined as *partial regenerative* if there exists a subset of the system state variables and the sub-system represented by the subset can repeat its state infinitely. The system is then partitioned at the *regeneration points* which trigger *regenerative sub-states*. In some cases, the regeneration points can be found without performing a detailed simulation [28, 29]; thus the state matching problem can be solved by performing a pre-computation. A *pre-simulation* to identify regenerative points by using Markovian modeling is discussed in [27].

The widespread use of time-parallel simulation is restricted by the fact that many systems of interest are not regenerative. Wireless networks fall in this category: a regeneration point would correspond to a state in which all the nodes turn of their transmitters and flush their routing tables and other state variables. Such states do not normally exist in simulation scenarios.

For systems which are not regenerative, instead of searching for exact results, we can look for approximate results which can be produced efficiently [24, 26, 30, 31]. A hard problem for any method proposing an approximate solution is to provide a reliable error estimation and an error control strategy.

In our previous work, we have adapted TPS to the simulation of wireless ad hoc networks [1–4]. We proposed a method which iteratively improves the accuracy of the simulation by extending the warmup interval. We implemented the method using the ns-2 simulator and used it to analyze the packet loss ratio and throughput for two popular ad hoc routing algorithms: AODV [5] and DSDV [6].

3 Compressed History

In the following, we describe a formal model of time parallel simulation of an ad hoc network and define the compressed history in the terms of this model.

A simulation S of an ad hoc network calculates the simulation trace $\mathcal{T}$ of event set E which occurs in geographic area A, within time interval τ, when the initial state is $\mathcal{I}$, and the final state is $\mathcal{F}$. Formally, we denote a simulation as a six-tuple $S = (A, \tau, E, \mathcal{I}, \mathcal{F}, \mathcal{T})$. We partition the temporal dimension of the simulation S into m time intervals $\{\tau_i | \tau_i = [(i-1)\frac{|\tau|}{m}, i\frac{|\tau|}{m}], 1 \leq i \leq m\}$, where $|\tau|$ is the duration of time interval τ.

A time-parallel simulation is a set of simulation segments $L = \{S_i | S_i = (A, \tau_i, E(S_i), \mathcal{I}(S_i), \mathcal{F}(S_i), \mathcal{T}(S_i)), 1 \leq i \leq m\}$. For any simulation segment S_i, the event set $E(S_i) \subset E$ is the events from E which fall in the time interval of S_i. Thus the segments of the time-parallel simulation simulate the full spatial component A, but only a subset of the temporal span τ_i. In order to have an exact simulation, the initial state of S_{i+1} needs to match the final state of S_i, $1 \leq i \leq m-1$. This final state, however, cannot be obtained without actually running all the previous simulation segments.

A crude approximation can be obtained assuming that every segment starts with an empty state, $L = \{S_i | S_i = (A, \tau_i, E(S_i), \emptyset, \mathcal{F}(S_i), \mathcal{T}(S_i)), 1 \leq i \leq m\}$ and combining the simulation traces $\mathcal{T}(L) = \mathcal{T}(S_1) \cup \mathcal{T}(S_2) \cup \ldots \cup \mathcal{T}(S_m)$ of the simulation segments. We call this raw time-parallel simulation method as TPS.

A better approximation can be obtained using a *warmup interval*. For every simulation segment

$$S_i = (A, [t_{i-1}, t_i], E(S_i), I(S_I), \mathcal{F}(S_i), \mathcal{T}(S_i))$$

we define a warmup simulation

$$S_{wi} = (A, [t_{i-1} - t_w, t_{i-1}], E(S_{wi}), \emptyset, \mathcal{F}(S_{wi}), \mathcal{T}(S_{wi}))$$

We will use the final state of the warmup interval as the initial state of the measured part of the simulation $\mathcal{F}(S_{wi}) = \mathcal{I}(S_i)$. We will call this simulation approach TPS-U (with the U denoting that the warmup interval is uncompressed). Naturally, the longer the warmup interval t_w, the better the approximation of the initial state. In the extreme case, $t_w = t_{i-1}$, and the warmup provides an exact value of the S_i. However, this would not provide any speedup over a linear simulation of the entire interval.

Let us now introduce the notion of the compressed history. Note that from the warmup simulation we are interested only of the final state $F(S_{wi})$. The computational effort to perform this simulation is monotonically increasing with the set of events in the warmup interval $E(S_{wi})$. Our goal is to find a different set of events $\hat{E}(S_{CHi})$, which, when replaces in the warmup simulation, will provide a final state which is either equal to the one of the full warmup $F(S_{wi}) = F(S_{CHi})$ or a close approximation of it $F(S_{wi}) \cong F(S_{CHi})$. Naturally, we are interested only in the cases where the simulation of $\hat{E}(S_{CHi})$ is significantly cheaper than that of $E(S_{wi})$. We define the *compression ratio* of the compressed history $\hat{E}(S_{CHi})$ by

$$\eta(\hat{E}_{CHi}, E_{wi}) = \frac{\text{executionTime}(S_{wi})}{\text{executionTime}(S_{CHi})}$$

In general, knowledge of the application domain, the perturbations caused by the specific classes of events, their interrelationships, as well as their impact on the simulation time can help us design compressed histories with high compression ratio, and high accuracy.

Let $\underline{S_i}$ be the complete sequential simulation prior to S_i, $\underline{S_i} = (A, \bigcup_{j=1}^{i-1}(\tau_j), E(\underline{S_i}), \emptyset, \mathcal{F}(\underline{S_i}), \mathcal{T}(\underline{S_i}))$, for $2 \leq i \leq m$. Now, the initial state of a simulation batch S_i can be approximated by the final state of the compressed history of its prior simulation $\underline{S_i}$, that is, $\underline{\hat{S_i}}$, for $2 \leq i \leq m$. Thus, we obtain a new time-parallel simulation set $L' = \{S'_i | S'_i = (A, \tau_i, E(S'_i), \mathcal{I}(S'_i), \mathcal{F}(S'_i), \mathcal{T}(S'_i)), 1 \leq i \leq m\}$, where

$$\mathcal{I}(S'_i) = \begin{cases} \emptyset, & \text{for } i = 1; \\ \mathcal{F}(\underline{\hat{S_i}}), & \text{for } 2 \leq i \leq m. \end{cases}$$

We call the time-parallel simulation algorithm with compressed history as TPS-C (see Figure 2).

Assume $S = (500 \times 500, [0, 180], E, \emptyset, \mathcal{F}, \mathcal{T})$ is segmented into 9 equal intervals, with $\tau_i = [20(i-1), 20i], 1 \leq i \leq 9$. The compressed history of the prior simulation for each simulation batch S_i ($2 \leq i \leq 9$) is: $(\underline{\hat{S_2}}, \underline{\hat{S_3}}, \ldots, \underline{\hat{S_9}})$.

The set of simulation batches without improvement is:

$$L = S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5 \cup S_6 \cup S_7 \cup S_8 \cup S_9,$$

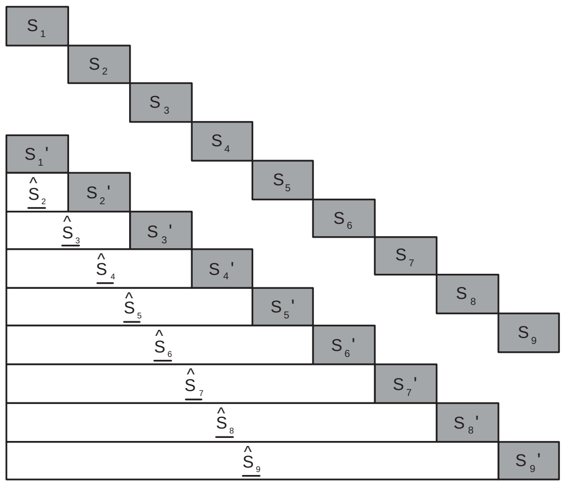


Figure 2: Comparison between algorithms TPS and TPS-C. The simulated time $|\tau| = 180$ sec. is segmented into 9 simulation intervals of 20 sec. each. The shaded simulation segments are used to compose the final simulation trace. $\{S_i\}$ is the set of simulation without initial state approximation, while $\{S_i'\}$ is the set of simulation batches with their initial states approximated by the compressed history of their prior simulation.

and the corresponding simulation result is

$$\mathcal{T}(L) = \mathcal{T}(S_1) \cup \mathcal{T}(S_2) \cup \mathcal{T}(S_3) \cup \ldots \cup \mathcal{T}(S_9).$$

The set of simulation batches with their initial states approximated by the compressed history of their prior simulation is

$$L' = S_1' \cup S_2' \cup S_3' \cup S_4' \cup S_5' \cup S_6' \cup S_7' \cup S_8' \cup S_9',$$

and the improved simulation result is

$$\mathcal{T}(L') = \mathcal{T}(S_1') \cup \mathcal{T}(S_2') \cup \mathcal{T}(S_3') \cup \ldots \cup \mathcal{T}(S_9').$$

Experimental results show that an aggressive compression of the warmup interval can produce high approximation errors. It turns out however, that the events at the very end of the compressed interval (that is, immediately before the start of the measured interval) are responsible for a large part of the approximation errors. One way to improve the accuracy is to introduce a short *uncompressed interval* between the compressed history and the beginning of the measured interval. The introduction of uncompressed simulation interval reduces the influence of perturbations at the measured point.

In the improved algorithm TPS-CU, the time interval $\underline{S_i}$ is split to two parts, the compressed history $\underline{\hat{S_i}}$ and the uncompressed interval U_i,

$$\underline{S_i} = \underline{\hat{S_i}} \cup U_i.$$

Note that the previously introduced TPS-C and TPS-U algorithms are variations of TPS-CU algorithm; TPS-C is the special case when $|\tau_c| = 0$, while TPS-U corresponds to $|\tau_u| = 0$.

Figure 3 illustrates the improved TPS algorithm for the same example. We fixed the duration of the uncompressed interval as 5 seconds for each simulation segment; the compressed history for simulation segment i is

$$\underline{\hat{S_i}} = (500 \times 500, [20(i-1), 20i - 5], E(\underline{\hat{S_i}}), \emptyset, \mathcal{F}(\underline{\hat{S_i}}), \mathcal{T}(\underline{\hat{S_i}})),$$

and the uncompressed simulation for simulation segment i is

$$U_i = (500 \times 500, [20i - 5, 20i], E(U_i), \mathcal{F}(\underline{\hat{S_i}}), \mathcal{F}(U_i), \mathcal{T}(U_i)).$$

Thus, we obtain a new time-parallel simulation set $L'' = \{S_i' | S_i' = (A, \tau_i, E(S_i'), \mathcal{I}(S_i'), \mathcal{F}(S_i'), \mathcal{T}(S_i')), 1 \leq i \leq m\}$, where

$$\mathcal{I}(S_i') = \begin{cases} \emptyset, & \text{for } i = 1; \\ \mathcal{F}(U_i), & \text{for } 2 \leq i \leq m. \end{cases}$$

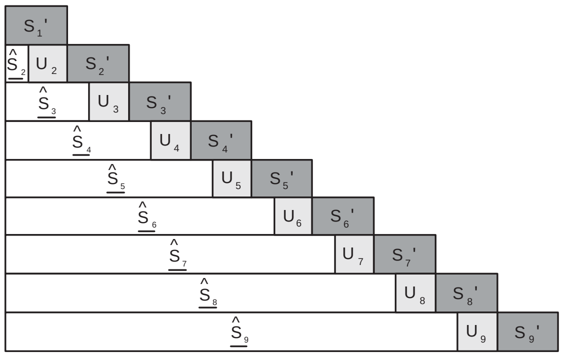


Figure 3: The TPS-CU (TPS with compressed history and uncompressed interval) algorithm . The $|\tau| = 180$ sec. interval is segmented into 9 intervals of 20 sec. each. The uncompressed interval duration is 5 sec. for each simulation segment. The prior simulation of S_i is composed of a compressed history $\underline{\hat{S_i}}$ and an uncompressed interval U_i. The dark shaded simulation segments are used to compose the final simulation trace. $\{S_i'\}$ is the set of simulation batches with their initial states approximated by the compressed history of their prior simulation and an uncompressed interval.

4 Implementation of Compressed History

After presenting the formalism of compressed history, in this section we turn to a concrete example and describe in detail the way in which history compression can be implemented in the case of a specific application. As we already mentioned in the introduction, the application we consider is ad hoc wireless networks using the Destination Sequenced Distance Vector (DSDV) [6] routing protocol.

In our previous work [1] using TPS with dynamically extended warmup we obtained speedups of 16-20 times for scenarios using AODV, but only 4-5 times for DSDV. The lower speedup is caused by the fact that a proactive protocol like DSDV maintains more state information (in the form of the routing tables), and thus requires longer and more expensive warmup for equivalent accuracy. Our goal is to replace part of the warmup interval with compressed history, such that the overall speedup will be brought closer to the reactive protocols.

To find an efficient compression mechanism, we need to identify a computationally expensive part of the protocol, which we can seek to compress. In DSDV, the nodes periodically update their routing tables even in the absence of any data flow (hence the proactive nature of the protocol). The justification for the frequent routing table updates is the mobility of the nodes. The routing table updates involve a significant number of packet exchanges, which are computationally expensive to simulate. Note, however, that for the measured interval, we are interested only in the final configuration of the routing table; the history of the entries does not affect the current state of the network. We say that the previous routing table updates are *shadowed* by the later ones.

Thus, one possible implementation of compressed history can be the selective removal of some of the routing table updates. We observe that the earlier is a routing table update in the history of the simulation, the more likely is that it will be shadowed by a future event. Thus, we can perform a compression of the history by reducing the frequency of the routing table updates. There is no guarantee that this method will yield a final state identical to the uncompressed history. It is possible, for instance, that a particularly long lived routing entry was created at a routing table update which was skipped, and never overwritten afterwards. On the other hand, there is a high likelihood that the missing entry will be, in fact, created at the next update, resulting in the same routing table at the beginning of the measured interval.

Figure 4 illustrates three possible strategies for adjusting the routing table update frequency. A sequential simulation is divided into a set of sub-intervals: in the *idle interval* no routing table update happens, in the *compressed history* the routing table updates are rarer than in the default protocol, while in the *measured interval* the routing table updates are regular. The vertical lines indicate the routing table update events, thus the density of the lines indicate the local frequency of routing table updates. With strategy 4(a), the frequency increases gradually in the compressed history interval, and the regular frequency is met right before the measured point. With strategy 4(b), the routing table updates sent long time ago are all ignored in the compressed history. Strategy 4(c) assumes that routing table updates happen at a constant interval during the compressed history, but less frequently than during the measured interval. Figure 5 illustrates the corresponding strategies with an uncompressed warmup interval inserted between the

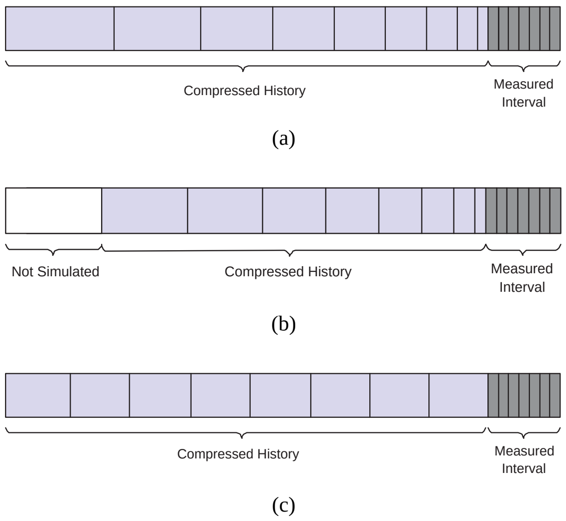


Figure 4: Strategies to adjust the routing table update frequency: (a) increase gradually the update frequency, (b) increase gradually the update frequency and ignore the updates older than a certain threshold and (c) keep the update frequency constant but lower than the update frequency in the measured interval.

compressed history and the measured interval. The routing table updates are sent at a regular level within the uncompressed interval. In our simulation, we implemented and investigated strategies 4(c) and 5(c). As the experimental results will show, even a small uncompressed interval can provide a significant improvement in the accuracy of the simulation.

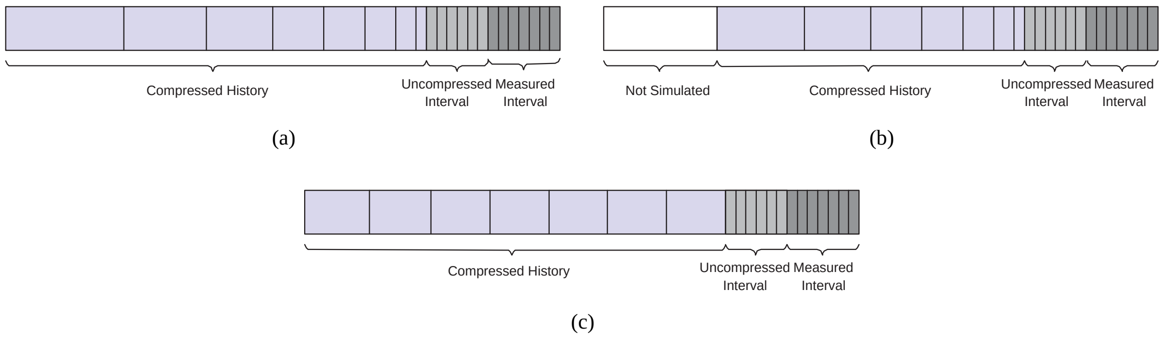


Figure 5: Strategies to adjust the routing table update frequency, with the application of uncompressed interval: (a) increase gradually the update frequency, (b) increase gradually the update frequency and ignore the updates older than a certain threshold and (c) keep the update frequency constant but lower than the update frequency in the measured interval. All three strategies insert an uncompressed interval before the measured interval.

4.1 Load balancing TPS with compressed history

The overall execution time of a parallel simulation is determined by the simulation segment with the longest execution time. As the measured interval of the individual time-parallel simulation threads starts at different time points, the length of the compressed history will be different. This leads to an imbalance in the execution time: with a constant compression ratio, the execution time of a simulation segment increases when its compressed history duration increases. Our goal is to balance the execution time of the compressed histories, by adjusting the compression ratio according to the compressed history duration.

Assume the routing table update frequency of a sequential simulation S_i is a constant λ_c, and the routing table update frequency of the compressed history $\underline{\hat{S_i}}$ is λ_i. The compression ratio $\eta \geq 1$ is the ratio of the original size of the history to the size of the compressed history. Thus a value of $\eta = 1$ represents no compression. The compression ratio of $\underline{\hat{S}_i}$ can be approximated by

$$\eta(\underline{\hat{S}_i}, S_i) \approx \frac{\lambda_c}{\lambda_i}.$$

Define the *normalized compressed history duration* as the compressed history duration divided by the compression ratio, which reflects the approximate duration of a compressed history if it was not compressed. Assume the compressed history duration of $\underline{\hat{S}_i}$ is $|\tau(\underline{\hat{S}_i})|$, the normalized compressed history duration is $|\tau(\underline{\hat{S}_i})| \,/\, \eta(\underline{\hat{S}_i}, S_i)$.

The balancing of the execution time of different simulation segments can be achieved by equalizing the normalized compressed history duration. If all simulation segments have a constant normalized compressed history duration $|\tau_c|$, that is

$$|\tau_c| = \frac{|\tau(\underline{\hat{S}_i})|}{\eta(\underline{\hat{S}_i}, S_i)} \approx |\tau(\underline{\hat{S}_i})| \cdot \frac{\lambda_i}{\lambda_c}$$

then the required routing table update frequency for each simulation segment can be obtained by

$$\lambda_i = \lambda_c \cdot \frac{|\tau_c|}{|\tau(\underline{\hat{S}_i})|}$$

4.2 Accuracy-speedup tradeoffs

In TPS-CU, the initial state of the measured interval S_i' is obtained as the final state of the uncompressed interval U_i, whose initial state is approximated by the final state of the compressed history $\underline{\hat{S}_i}$.

Thus, we have two ways to increase the accuracy of the initial state in the TPS-CU algorithm: either we increase the normalized compressed history duration of $\underline{\hat{S}_i}$ (that is, decrease the compression ratio), or we increase the length of the uncompressed history duration. Unfortunately, this way we also reduce the speedup of the simulation.

Let us consider a simulation S over a duration τ, segmented into m equal measured intervals. The normalized compressed history ($\underline{\hat{S}_i}$) duration is $|\tau_c|$, and the uncompressed interval (U_i) duration is $|\tau_u|$. Let $t(S)$ be the function that calculates the execution of a simulation S. The speedup of our algorithm is

$$\kappa = \frac{t(S)}{\max_i\{t(\underline{\hat{S}_i}) + t(U_i) + t(S_i')\}}.$$

If we assume that the events are evenly distributed over the simulation interval and the execution of each event takes about the same time Δt then

$$t(S) \approx |E(S)| \cdot \Delta t$$

If we ignore the execution time for data packets, the speedup can be approximated by

$$\begin{aligned}
\kappa &= \frac{t(S)}{\max_i\{t(\underline{\hat{S}_i}) + t(U_i) + t(S_i')\}} \\
&\approx \frac{|E(S)| \cdot \Delta t}{\max_i\{(|E(\underline{\hat{S}_i})| + |E(U_i)| + |E(S_i')|) \cdot \Delta t\}} \\
&= \frac{\lambda_c \cdot |\tau|}{\max_i\{\lambda_i \cdot |\tau(\underline{\hat{S}_i})| + \lambda_c \cdot |\tau_u| + \lambda_c \cdot |\tau_i|\}} \\
&= \frac{\lambda_c \cdot |\tau|}{\lambda_c \cdot |\tau_c| + \lambda_c \cdot |\tau_u| + \lambda_c \cdot \dfrac{|\tau|}{m}} \\
&= \frac{|\tau|}{|\tau_c| + |\tau_u| + \dfrac{|\tau|}{m}}.
\end{aligned}$$

According to this formula, there are three possible means to increase the speedup: (1) decrease the normalized compressed history duration $|\tau_c|$; (2) decrease the uncompressed interval duration $|\tau_u|$; or (3) increase the number of measured intervals m, which is equivalent to decreasing the duration of the individual measured intervals. Each of these choices also affects the accuracy of the simulation. Our goal is to determine the optimal values of: (i) the normalized compressed history duration, (ii) the uncompressed interval duration, and (iii) the measured interval duration such that we achieve the largest possible speedup while meeting accuracy constraints.

5 A Simulation Study

In this section we describe the results of a simulation study which studies the accuracy of time-parallel simulation with compressed history. We compare the approximate results produced by TPS-U, TPS-C and TPS-CU with those produced by exact sequential simulation over several frequently used metrics.

Table 1: Abbreviation and notations

TPS-C	TPS with compressed history
TPS-U	TPS with uncompressed interval
TPS-CU	TPS with compressed history and uncompressed interval
$\mid \tau_i \mid$	the duration of segment i
$\mid \tau_u \mid$	the duration of the uncompressed interval
$\mid \tau_c \mid$	the duration of the normalized compressed history
$\varepsilon(P)$	the relative error for packet loss ratio
$\varepsilon(T)$	the relative error for the throughput
ε_{max}	the relative error threshold
κ	the speedup of the algorithm

Table 1 lists the abbreviations and notations used throughout this section. The simulation scenarios on which we test our implementations are representative for the scenarios used in the study of ad hoc wireless networks. A set of mobile nodes are moving in a rectangular area, while communicating using wireless connections. The transmission range of the nodes is smaller than the size of the area, and there are no fixed wireless access points available. Thus, in order to communicate, the nodes need to use hop-by-hop transmissions relying on ad hoc routing algorithms. Due to the mobility of the nodes, the routes need to be frequently updated, even during the course of the transmissions.

In our experimental study, 80 nodes move in a simulation area of size 500 $\times$ 500; all nodes have a transmission range of 100 meters. The duration of the scenario is 600 seconds. The node movement is modeled by the random waypoint model [32, 33]. The traffic patterns are generated by constant bit rate (CBR) sources sending 512-byte UDP packets at a rate of 1 packet/second. The nodes use the Destination-Sequenced Distance Vector (DSDV) routing algorithm. Table 2 shows the default settings of the parameters for our experiments.

Table 2: The default values of the parameters

Field	**Value**
simulation area	$500 \times 500(m^2)$
number of nodes	80
transmission range	100 (m)
speed	1 (m/s)
pause time	15 (s)
simulation time	600 (s)
number of CBR sources	20
CBR packet size	512 (bytes)
CBR sending rate	4 (kbps)

Let us now describe the specific approaches we compare.

- **TPS-U** is time-parallel simulation with uncompressed warmup interval. This is the baseline algorithm described in [1].
- **TPS-C** is time-parallel simulation with the warmup interval being composed exclusively of compressed history. The strategy used for this approach is the one described in Figure 4(c), that is, the warmup period extends to the complete simulation scenario, and the routing table update frequency is equidistant. To balance the load of the simulation threads, we maintained a constant normalized compressed history duration.
- **TPS-CU** is time-parallel simulation with a warmup interval composed of a compressed interval followed by an uncompressed interval. The particular strategy used in this simulation study is the one described in Figure 5(c), that is, the compressed interval starts at the beginning of the simulation scenario and the routing table

update frequency is constant for a given simulation. To balance the load of the simulation threads, we maintained an uncompressed component of constant duration and a compressed component of constant normalized compressed history duration.

Our approach for comparing the accuracy of different simulation runs is based on the calculation of the relative error of specific measurement types. First, we run a serial, accurate simulation and extract an exact measurement M_0 on its trace. Then, we perform a time-parallel simulation with one of the proposed algorithms (TPS-U, TPS-C and TPS-CU), assemble the simulation trace from the segments, and perform the same measurement, obtaining an approximate value M. The relative error of the specific algorithm for this scenario will be $\varepsilon(M) = \frac{M-M_0}{M_0} \times 100\%$. As the relative error fluctuates from scenario to scenario, we repeat the experiment 20 times, each time with a different random configuration of the source destination pairs of the CBR sources. We calculate the mean, as well as the 95% confidence interval of the relative error. The two measurements considered in this study are the relative error for the packet loss ratio $\varepsilon(P)$ and the relative error for throughput $\varepsilon(T)$.

The relative error of simulation tends to be larger for metrics which measure rare phenomena (such as packet loss), compared to metrics which measure frequently occurring phenomena (such as throughput) [1]. As a result, $\varepsilon(T)$ is always smaller than $\varepsilon(P)$. The error threshold ε_{max} used in this study was 5%. In practice, this means that the measurement of the packet loss ratio will be roughly 95% accurate, while the accuracy of the measured throughput will be much higher, around 99.5% for most simulation runs.

The speedup of the algorithm was determined using the method discussed in Section 4.2.

The time-parallel simulation was run on a cluster of 128 64-bit Opteron processors with Gigabit Ethernet interconnection. Each simulation segment was run on its own dedicated processor.

5.1 Tuning the parameters of the compressed history for optimal speedup

The TPS-C and TPS-CU algorithms have several parameters that can determine the speedup and accuracy of the algorithm. Before we compare the accuracy and the speedup of the TPS-U, TPS-C and TPS-CU algorithms, we need to determine the parameters under which the compressed history approach works best. These tuned parameters will be then used in the actual comparisons. Note that this tuning process does not need to be done before every simulation; the parameters are specific to the protocol simulated and the compressed history implementation. The developer implementing the compressed history would normally perform this tuning, and present the optimal parameters in the form of easy to use rules-of-thumb.

The most general case is TPS-CU with three parameters: the segment duration $|\tau_i|$ (which is inversely proportional with the number of simulation threads), the duration of the normalized compressed history $|\tau_c|$ and the duration of the uncompressed interval $|\tau_u|$.

We allow these parameters to vary over a range of values: $10 \leq |\tau_i| \leq 40$, $0 \leq |\tau_c| \leq 40$ and $0 \leq |\tau_u| \leq 40$, and retain the values that yield the highest speedup for a target accuracy.

Figures 6 and 7 show the relative error for packet loss ratio and the throughput function of the duration of the uncompressed interval, for several segment lengths. Given a segment and a normalized compressed history of fixed length, the relative error for both metrics tend to decrease when the uncompressed interval duration increases. Also, the relative error for the two metrics tend to decrease when the normalized compressed history duration increases, given the duration of the segment and of the uncompressed interval. The relative errors for the two metrics decrease when the segment duration increases. For instance, with $(|\tau_u|, |\tau_c|) = (5, 5)$ seconds, the relative error for packet loss ratio is reduced from 39.76% when $|\tau_i| = 10$ seconds, to 24.09% when $|\tau_i| = 40$ seconds; the relative error for the throughput is reduced from 8.82% when $|\tau_i| = 10$ seconds, to 3.90% when $|\tau_i| = 40$ seconds.

To maximize the speedup for a given $|\tau_u|$, we choose the minimal $|\tau_c|$ such that the relative errors for the corresponding metrics are below ε_{max}. We conducted a series of simulations with different segment durations. The simulation results are summarized in Tables 3 (a), (b), (c), and (d). Each table shows the minimal value of $|\tau_c|$, the normalized compressed history duration as well as $\varepsilon(P)$, the relative error for the packet loss ratio, $\varepsilon(T)$, the relative error for the throughput, and κ, the speedup, function of τ_u, the uncompressed interval duration. The normalized compressed history duration required to achieve the accuracy threshold tends to decrease when the uncompressed interval duration increases. When $|\tau_i| = 10$ sec., the largest speedup is $\kappa = 6.51$ with $(|\tau_u|, |\tau_c|) = (35, 10)$ seconds and the relative errors are $(\varepsilon(P), \varepsilon(T)) = (4.63\%, 0.63\%)$; when $|\tau_i| = 40$ sec., the largest speedup is 6.79 with $(|\tau_u|, |\tau_c|) = (15, 15)$ seconds and the relative errors are $(\varepsilon(P), \varepsilon(T)) = (4.07\%, 0.50\%)$.

Table 4 lists the combination of the uncompressed interval duration and the normalized compressed history duration that led to the largest speedup, under all values of segment duration that is simulated. The largest speedup

Table 3: Tuning the parameters of the TPS-CU algorithm based on the results of the experimental runs. The rows corresponding to the best choice of the uncompressed interval $|\tau_u|$ for a given segment duration are shown in bold. The maximum achievable speedup for a given segment duration is bold and underlined.

$\lvert\tau_u\rvert$	min $\lvert\tau_c\rvert$	$\varepsilon(P)$	$\varepsilon(T)$	κ
10	40	4.42%	0.59%	4.01
15	30	4.15%	0.57%	5.29
20	25	4.70%	0.67%	5.30
25	25	3.15%	0.41%	5.20
30	15	3.32%	0.45%	6.32
35	**10**	**4.63%**	**0.63%**	**<u>6.51</u>**
40	5	4.31%	0.58%	5.67

(a) Segment Duration = 10 seconds

$\lvert\tau_u\rvert$	min $\lvert\tau_c\rvert$	$\varepsilon(P)$	$\varepsilon(T)$	κ
10	35	4.32%	0.53%	6.62
15	30	4.13%	0.49%	6.33
20	**20**	**3.94%**	**0.48%**	**<u>7.34</u>**
25	20	4.21%	0.51%	6.43
30	10	4.80%	0.59%	7.30
35	10	4.29%	0.51%	6.39
40	5	4.73%	0.58%	6.39

(b) Segment Duration = 20 seconds

$\lvert\tau_u\rvert$	min $\lvert\tau_c\rvert$	$\varepsilon(P)$	$\varepsilon(T)$	κ
5	35	3.76%	0.51%	4.88
10	25	4.77%	0.65%	7.44
15	15	4.01%	0.54%	7.46
20	**15**	**3.70%**	**0.49%**	**<u>7.54</u>**
25	10	4.12%	0.55%	6.99
30	5	4.64%	0.63%	6.84

(c) Segment Duration = 30 seconds

$\lvert\tau_u\rvert$	min $\lvert\tau_c\rvert$	$\varepsilon(P)$	$\varepsilon(T)$	κ
5	30	3.70%	0.66%	5.51
10	25	4.59%	0.56%	6.17
15	**15**	**4.07%**	**0.50%**	**<u>6.79</u>**
20	15	4.61%	0.58%	6.04
25	15	3.33%	0.32%	6.43
30	5	3.86%	0.46%	6.14

(d) Segment Duration = 40 seconds

increases when the segment duration increases from 10 to 30 sec., and starts to decrease when the segment duration increases beyond 30 sec. The largest speedup, $\kappa = 7.54$, is achieved when $(|\tau_i|, |\tau_u|, |\tau_c|) = (30, 20, 15)$ sec. These values will be chosen for the simulation results presented later.

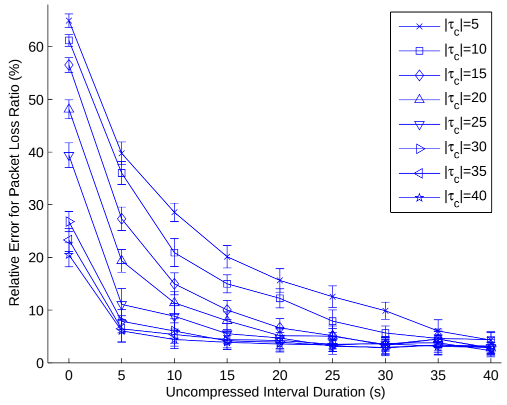
Table 4: The combination of uncompressed interval duration and normalized compressed history duration that achieve maximum speedup for different segment durations

$\lvert\tau_i\rvert$	$\lvert\tau_u\rvert$	$\lvert\tau_c\rvert$	κ
10	35	10	6.51
20	20	20	7.34
30	**20**	**15**	**7.54**
40	15	15	6.79

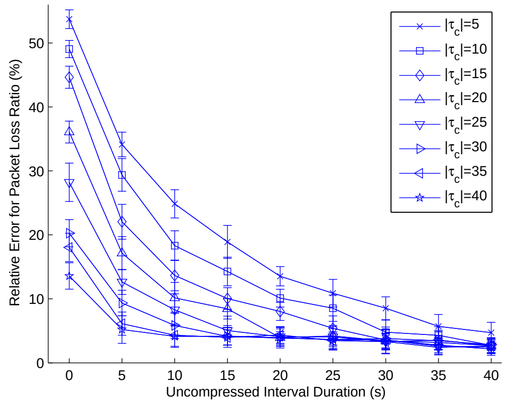
Before we proceed further, let us consider the costs and benefits of tuning.

The benefits of tuning can be inferred from Table 4. The best choice of parameters can provide more that 50% additional speedup compared to the worst choice. One the other hand, getting the parameters "slightly wrong" comes with only a minor penalty.

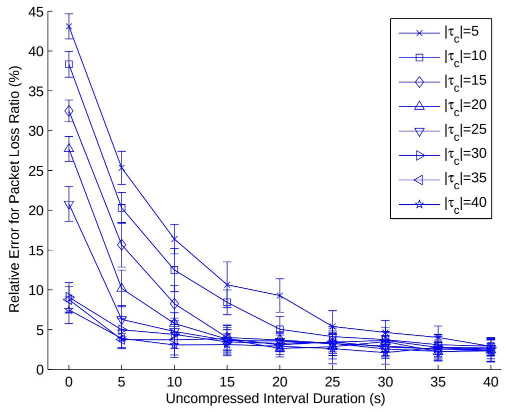
However, for any simulation for which time parallel simulation is justified, the cost of tuning will be significant, as it requires extensive parameter sweeps over a range of scenarios. The results in Table 4 and 8 were obtained through more than 2 weeks of exclusive computer time on a 128 node cluster! Clearly, it is not worth for the user to perform a tuning process in order to run a single simulation. However, the cost of tuning is small compared to the cost of developing a compressed history implementation. The developer of the compressed history implementation can perform an initial tuning as part of the validation of the implementation. Subsequently, the users can use a rule-of-thumb table, such as Table 8, to pick the best parameters for a specific deployment.



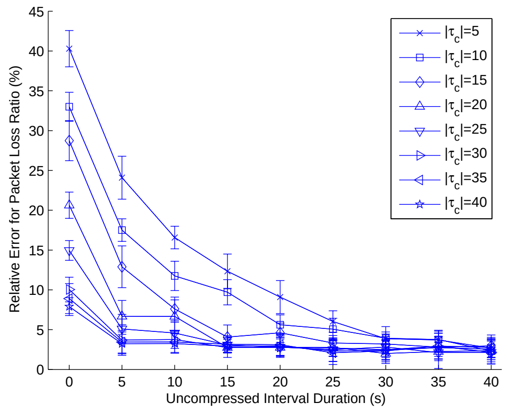
(a) Segment Duration = 10 seconds



(b) Segment Duration = 20 seconds



(c) Segment Duration = 30 seconds



(d) Segment Duration = 40 seconds

Figure 6: The TPS-CU algorithm: the relative error for packet loss ratio function of uncompressed interval duration. The error decreases when we increase either the uncompressed interval duration, or the normalized history duration, or the segment duration.

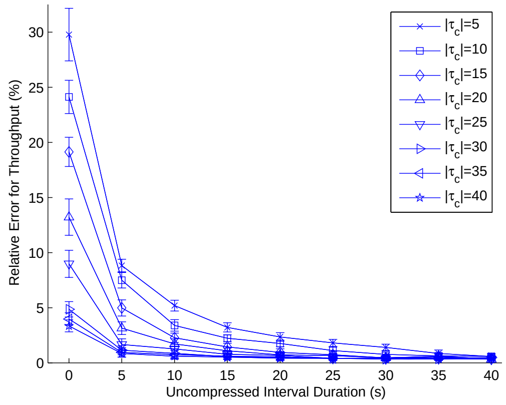
5.2 The performance improvement provided by compressed history: comparing TPS-CU with TPS-U

We investigate the effect of the compressed history, for a range of durations of the uncompressed interval when the segment duration is 30 sec., the normalized compressed history duration for the TPS-CU algorithm is 15 sec., and the uncompressed interval duration is $10 \leq |\tau_u| \leq 40$ sec.

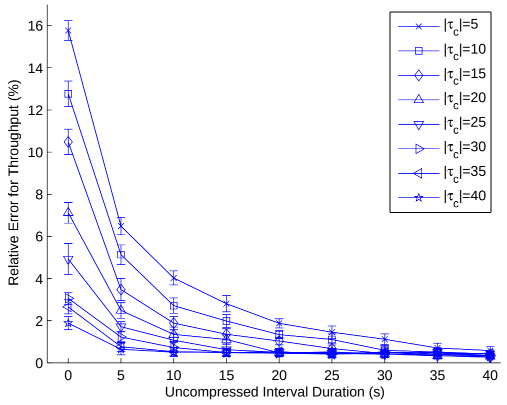
Figure 8 shows the relative errors produced by the TPS-U and TPS-CU algorithms. For both algorithms the relative error for packet loss ratio and the throughput tend to decrease when the uncompressed interval duration increases. For the same duration of the uncompressed interval the relative errors produced by the TPS-CU are smaller than the ones produced by the TPS-C algorithm; the TPS-U algorithm produces errors below ε_{max} for considerably smaller values of the uncompressed interval.

Table 5 summarizes the effect of the compressed interval upon the relative errors. A 15-sec. normalized compressed history, leads to the reduction of the relative error for packet loss ratio from 69.42% to 8.26% when $|\tau_u| = 10$ seconds and from 43.20% to 2.58% when $|\tau_u| = 30$ seconds; the relative error for the throughput is reduced from 38.84% to 1.39% when $|\tau_u| = 10$ seconds, and from 10.38% to 0.50% when $|\tau_u| = 30$ seconds.

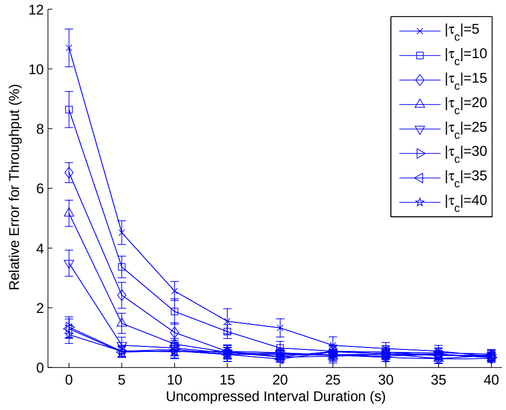
We conclude that the compressed history allows us to obtain a better approximation for both the packet loss ratio and the throughput.



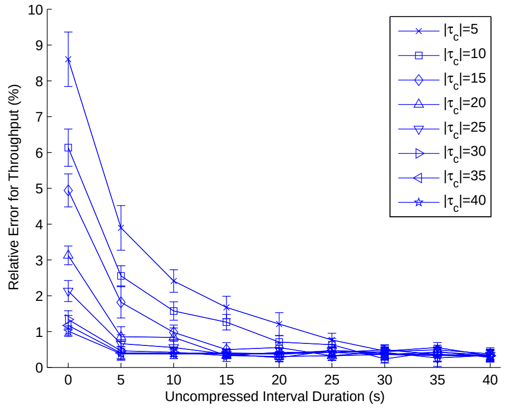
(a) Segment Duration = 10 seconds



(b) Segment Duration = 20 seconds



(c) Segment Duration = 30 seconds



(d) Segment Duration = 40 seconds

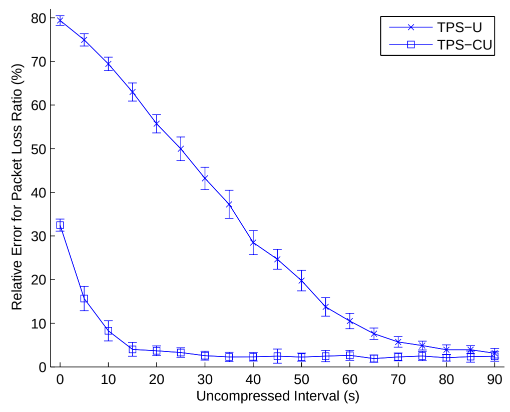
Figure 7: The TPS-CU algorithm: the relative error for the throughput function of the duration of the uncompressed interval. The error decreases when we increase either the uncompressed interval duration, the normalized history duration, or the segment duration.

Table 5: The effect of the compressed history upon the relative error for packet loss ratio and the throughput.

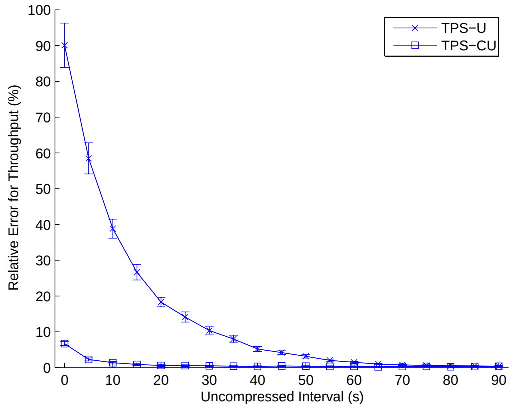
$\lvert\tau_u\rvert$	$\varepsilon(P)$		$\varepsilon(T)$	
	$\lvert\tau_c\rvert = 0$	$\lvert\tau_c\rvert = 15$	$\lvert\tau_c\rvert = 0$	$\lvert\tau_c\rvert = 15$
0	79.38%	**32.48%**	90.10%	**6.70%**
10	69.42%	**8.26%**	38.84%	**1.39%**
20	55.71%	**3.70%**	18.32%	**0.59%**
30	43.20%	**2.58%**	10.38%	**0.50%**
40	28.49%	**2.31%**	5.21%	**0.34%**
50	19.77 %	**2.24%**	3.15%	**0.40%**
60	10.49%	**2.64%**	1.47%	**0.34%**
70	5.72%	**2.28%**	0.75%	**0.31%**
80	3.93%	**2.09%**	0.50%	**0.25%**

5.3 The performance improvement provided by the uncompressed interval: comparing TPS-CU with TPS-C

The next question we investigate is whether we can dispense the uncompressed interval entirely and just rely on the compressed history to provide the warmup, as in the case of the TPS-C algorithm. We compared the TPS-CU and

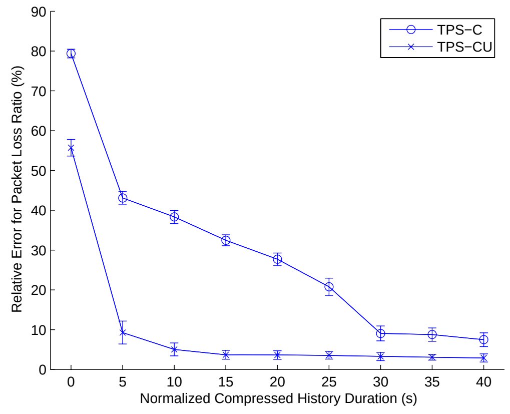


(a)

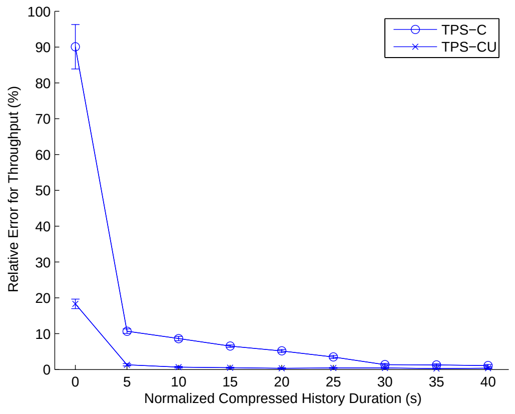


(b)

Figure 8: TPS-U and TPS-CU algorithms; the relative error function of the duration of the uncompressed interval for the: (a) packet loss ratio and (b) the throughput, when the segment duration is 30 sec., and the normalized compressed history duration for TPS-CU is 15 sec.



(a)



(b)

Figure 9: The TPS-C and TPS-CU algorithms; the relative error function of the duration of the uncompressed interval for (a) the packet loss ratio, and (b) the throughput, when the segment duration is 30 sec., and the uncompressed interval duration for TPS-CU is 20 sec.

TPS-C algorithms over a variety of values of normalized compressed history duration. In the experiments, the segment duration is set to 30 seconds and the uncompressed interval duration for TPS-CU is set to 20 seconds, based on the parameter tuning experiments performed previously. We vary the normalized compressed history duration $|\tau_c|$ from 0 to 40 seconds. TPS-U is represented by the scenario when $|\tau_u| = 0$.

Figure 9 compares the performance of the two algorithms, as function of normalized compressed history duration. The relative errors for packet loss ratio and the throughput tend to decrease when the normalized compressed history duration increases, for both algorithms. The relative error for the TPS-CU algorithm is consistently lower than for TPS-U, for a given the normalized compressed history duration. We find that the TPS-C algorithm requires significantly larger segment durations than TPS-CU to maintain the relative error below ε_{max}.

Table 6 summarizes the effect of the uncompressed interval. A 20-second uncompressed interval leads to a reduction of the relative error for packet loss ratio from 43.09% to 9.29% for $|\tau_c| = 5$ sec., and from 32.48% to 3.70% for $|\tau_c| = 15$ sec.; the relative error for the throughput is reduced from 10.70% to 1.32% for $|\tau_c| = 5$ sec., and from 6.53% to 0.49% for $|\tau_c| = 15$ sec.

Table 6: The effect of the uncompressed interval upon the relative error for packet loss ratio and throughput.

$\lvert\tau_c\rvert$	$\varepsilon(P)$		$\varepsilon(T)$	
	$\lvert\tau_u\rvert = 0$	$\lvert\tau_u\rvert = 20$	$\lvert\tau_u\rvert = 0$	$\lvert\tau_u\rvert = 20$
0	79.37%	**55.71%**	90.10%	**18.32%**
5	43.09%	**9.29%**	10.70%	**1.32%**
10	38.33%	**5.05%**	8.64%	**0.65%**
15	32.48%	**3.70%**	6.53%	**0.49%**
20	27.71%	**3.64%**	5.16%	**0.35%**
25	20.79%	**3.57%**	3.49%	**0.43%**
30	9.07%	**3.28%**	1.35%	**0.43%**
35	8.77%	**3.09%**	1.29%	**0.28%**
40	7.48%	**2.91%**	1.10%	**0.36%**

We conclude that the uncompressed interval in the TPS-CU algorithm provides a significant improvement over the TPS-C algorithm which relies exclusively on compressed history warmup.

6 Notes on generality

This paper proposes compressed history as a methodology to speed up time parallel simulation. We have used the simulation of the ad hoc wireless networking protocol DSDV to show some of the techniques through which compressed history can be implemented. We have obtained significant speedup in this particular case.

Naturally, however, the reader might ask the question: but it is general? What results can I expect if I try to implement or use compressed history in a different circumstance than the experiments presented here? Do the results change under different applications or scenarios, or if we are measuring different metrics?

While many of these questions represent future research topics, in the following we outline some considerations.

Application dependence – implementation. The implementation of the compressed history depends on the simulated process, as the compression needs to be applied to the state maintained by the process. The results obtained in this paper were based on a compression technique specific for DSDV. If we are trying to simulate a different network protocol, appropriate compression methods need to be written. Note, however, that this is not different from the fact specific implementation needs to be provided for the serial simulation as well. The best choice is to implement the compressed history as part of the basic simulation code.

Although the implementation is application dependent, there are some standard ways to implement compression:

- Eliminating shadowed events, whose effects have been overwritten by later events.
- Eliminating obsolete events, which have effects limited in time, which are too old to effect the state at the beginning of the simulation).
- Reducing trains of repetitive events to a smaller number of representatives.

Application dependence – performance. The benefits of the compressed history approach are also application dependent. If the simulated process maintains little or no history, the benefits of compressed history are minimal – but such processes require only a very small warmup period to reach acceptable accuracy. If the application maintains a long and complex history, it requires a very long warmup, and the benefits of compressed history can be substantial, as long as an efficient compression method can be found. Naturally, one can imagine processes where there is no opportunity for compression. For instance, a process where every event changes all the previous state variables and adds a new one would not have shadowed events, obsolete events or trains of repetitive events and thus would be a process which does not have shadowed events, and thus would not be compressible. However, most real world processes do not fall under this category.

Implementation dependency. The results of the TPS with compressed history does not depend on the implementation of the simulation for the considered application. Every correct implementation should yield the same result. When implementing the simulation of a new protocol, the developers might find it convenient to integrate the simulation proper and the implementation of compressed history, but this is not strictly necessary.

Scenario dependency. The results of compressed history can depend on the scenario, if the simulated processes reliance on history varies from scenario to scenario. For the particular case of networking protocols, our experience in [4] shows that the accuracy of TPS shows only a weak dependency on the scenario parameters. In most cases, the compressibility of the history is a fundamental feature of the process and it will not change with the scenarios. This allows us to tune the simulation parameters once and maintain a good confidence that future simulations will converge in a similar way as long as we maintain the parameters in the range in which the tuning was performed.

One can build (somewhat artificial) examples where the variation with the scenario is large: for instance a system which uses proactive routing for high traffic situations, but switches to reactive routing when the traffic is low. This, however, would be more an application change than a scenario change.

Dependency on the analysis metric. Just like any simulation in general, the accuracy of TPS with compressed history is dependent on the metric of interest. In general, the accuracy is much better for aggregate metrics summarizing a large number of events, as opposed to metrics which measure rare events. This phenomena, however, is not specific to the accuracy gap between TPS with compressed history to the corresponding serial simulation, but also to the relationship of the computer model of a process and its counterpart in the physical reality.

Note that in our experimental studies, we reported the speedups for the simulation length where the target accuracy of 5% was reached for the more problematic metric, the packet loss. For the aggregate metric of throughput the simulation reaches the target accuracy an order of magnitude faster. The reason for this is that in a normal scenario successful transmission is a relatively common event, while packet loss is comparatively rare.

7 Conclusions and Future Work

In this paper, we introduced compressed history, a method to improve the speedup of time parallel simulation with dynamically extended warmup. We have demonstrated the benefits of compressed history by applying it to the simulation of DSDV, a proactive routing protocol for wireless ad hoc networks.

Through a simulation study, we find that the use of compressed history provides a significant improvement in the speedup compared to the time parallel simulation with dynamically extended warmup, reported in [1–4]. The best results were obtained by combining compressed history with a short uncompressed warmup interval.

The compressed history approach opens a wide range of future research topics. The first, natural extension is to implement compressed history to other ad hoc networking protocols. The primary target would be the proactive and hybrid protocols, as reactive protocols, in general, carry less history and thus offer less opportunity for compression.

A second direction is in finding new compression methods. Our current implementation is eliminating shadowed routing protocol updates, but it does not touch the application data flow. Additional compression opportunities might be found by using the knowledge of node mobility.

Finally, we plan to apply the compressed history approach to application domains outside the simulation of ad hoc wireless networking protocols, potentially through interdisciplinary collaborations.

Acknowledgments

The research reported in this paper was partially supported by National Science Foundation grants ACI0296035 and EIA0296179 and by the Walton award from the Science Foundation of Ireland to one of the authors (dcm).

References

[1] L. Bölöni, D. Turgut, G. Wang, D. Marinescu, Challenges and benefits of time-parallel simulation of wireless ad hoc networks, in: Proc. 1st Int. Conf. on Perform. Evaluation Methodologies and Tools (Valuetools-2006), 2006.

[2] D. Turgut, G. Wang, L. Bölöni, D. Marinescu, Speedup-precision tradeoffs in time-parallel simulation of wireless ad hoc networks, in: Proc. 10th IEEE Int. Symp. on Distrib. Simul. and Real Time Appl. (DS-RT), 2006, pp. 265–268.

[3] G. Wang, D. Turgut, L. Bölöni, D. Marinescu, Accuracy-speedup tradeoffs for a time-parallel simulation of wireless ad hoc networks, in: Accepted for publication in Proc. 2nd IEEE Int. Workshop on Perform. and Management of Wireless and Mobile Networks (P2MNet), 2006.

[4] G. Wang, D. Turgut, L. Bölöni, D. Marinescu, Time-parallel simulation of wireless ad hoc networks, Accepted for publication at ACM/Springer Journal of Wireless Networks (WINET).

[5] C. E. Perkins, E. M. Royer, Ad hoc On-demand Distance Vector Routing, in: Proc. 2nd IEEE Workshop on Mobile Comput. Syst. and Appl., 1999, pp. 99–100.

[6] C. E. Perkins, P. Bhagwat, Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers, in: ACM Spec. Interest Group on Data Commun. (ACM SIGCOMM), 1994, pp. 234–244.

[7] R. M. Fujimoto, Parallel discrete event simulation, Commun. of ACM 33 (10) (1990) 30–53.

[8] R. M. Fujimoto, Parallel and distributed discrete event simulation: algorithms and applications, in: WSC '93: Proc. 25th Conf. on Winter simul., ACM Press, 1993, pp. 106–114.

[9] H. T. Mouftah, R. P. Sturgeon, Distributed discrete event simulation for communications networks, IEEE J. on Selected Areas in Commun. 8 (9) (1990) 1723–1734.

[10] A. Ferscha, S. K. Tripathi, Parallel and distributed simulation of discrete event systems, Tech. Rep. CS-TR-3336, Univ. of Maryland (AUGUST 1994).

[11] Y.-B. Lin, P. A. Fishwick, Asynchronous parallel discrete event simulation, IEEE Trans. Syst., Man and Cybernatics 26 (4).

[12] K. M. Chandy, J. Misra, Asynchronous distributed simulation via a sequence of parallel computations, Commun. ACM 24 (4) (1981) 198–206.

[13] R. E. Bryant, Simulation of packet communication architecture computer systems, Tech. rep., Massachusetts Institute of Technology, Cambridge, MA, USA (1977).

[14] A. Boukerche, C. Tropper, A static partitioning and mapping algorithm for conservative parallel simulations, SIGSIM Simul. Dig. 24 (1) (1994) 164–172.

[15] A. Boukerche, S. K. Das, Dynamic load balancing strategies for conservative parallel simulations, in: 11th Workshop on Parallel and Distributed Simulation (PADS'97), 1997, pp. 20–28.

[16] D. R. Jefferson, Virtual time, ACM Trans. Program. Lang. Syst. 7 (3) (1985) 404–425.

[17] R. M. Fujimoto, Lookahead in parallel discrete-event simulation, in: Proc. of 1988 International Conference on Parallel Processing, 1988, pp. 34–41.

[18] A. Boukerche, S. K. Das, A. Fabbri, O. Yildiz, Exploiting model independence for parallel PCS network simulation, in: 13th Workshop on Parallel and Distributed Simulation (PADS'99), 1999, pp. 166–173.

[19] A. Boukerche, S. K. Das, A. Fabbri, Swimnet: A scalable parallel simulation testbed for wireless and mobile networks, Wireless Networks 7 (5) (2001) 467–486.

[20] PDNS – Parallel/Distributed NS, URL `http://www.cc.gatech.edu/computing/compass/pdns/` (2004).

[21] VINT, The UCB/LBNL/VINT network simulator-ns (version 2), URL `http://www.isi.edu/nsnam/ns`.

[22] S. Andradóttir, T. J. Ott, Time segmentation parallel simulation of networks of queues with loss or communication blocking, ACM Trans. Model. and Comput. Simul. 5 (4) (1995) 269–305.

[23] M. Hoseyni-Nasab, S. Andradóttir, Parallel simulation by time segmentation: Methodology and applications, in: Proc. 1996 Winter Simul. Conf., 1996, pp. 376–381.

[24] M. Hoseyni-Nasab, S. Andradóttir, Time segmentation parallel simulation of tandem queues with manufacturing blocking, in: Proc. 1998 Winter Simul. Conf., 1998.

[25] Y. Lin, E. D. Lazowska, A time-division algorithm for parallel simulation, ACM Trans. Model. and Comput. Simul. 1 (1) (1991) 73–83.

[26] J. J. Wang, M. Abrams, Approximate time-parallel simulation of queueing systems with losses, in: Proc. 24th Conf. on Winter simul. (WSC ’92), ACM Press, 1992, pp. 700–708.

[27] J. J. Wang, M. Abrams, Determining initial states for time-parallel simulations, in: Proc. 7th workshop on Parallel and distrib. simul. (PADS ’93), ACM Press, 1993, pp. 19–26.

[28] I. Nikolaidis, R. Fujimoto, C. A. Cooper, Parallel simulation of high-speed network multiplexers, IEEE Conf. on Decision and Control 3 (1) (1993) 2224–2229.

[29] I. Nikolaidis, R. Fujimoto, C. A. Cooper, Time-parallel simulation of cascaded statistical multiplexers, in: Proc. 1994 ACM SIGMETRICS Conf. on meas. and model. of comput. syst., 1994, pp. 231–240.

[30] T. Kiesling, Approximate time-parallel cache simulation, in: Proc. 2004 Winter Simul. Conf., 2004, pp. 345–354.

[31] T. Kiesling, Using approximation with time-parallel simulation, Simulation 81 (4) (2005) 255–266.

[32] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, in: Proc. 4th Annual ACM/IEEE Int. Conf. on Mobile Comput. and Networking (MOBICOM ’98), 1998, pp. 85–97.

[33] D. B. Johnson, D. A. Maltz, Dynamic source routing in ad hoc wireless networks, in: Imielinski, Korth (Eds.), Mobile Computing, Kluwer Academic Publishers, 1996, pp. 153–181.