Time-Parallel Simulation of Wireless Ad Hoc Networks

Guoqiang Wang, Damla Turgut Ladislau Bölöni, and Dan C. Marinescu

School of Electrical Engineering and Computer Science
University of Central Florida
Orlando FL 32816, USA
{gwang,turgut,lboloni,dcm}@eecs.ucf.edu

Abstract. In this paper, we study time-parallel simulation of wireless networks based upon the concept of the perturbation induced by a networking event and present a layer-by-layer analysis of the impact of perturbations on the wireless network. This analysis allows us to propose several methods to improve the accuracy of time-parallel simulation. We describe an implementation based on the widely used ns-2 simulator and on the iterative extension of the warmup period. We introduce a method for initial state approximation which can improve the accuracy of the simulation for table-driven ad hoc routing protocols. A series of experiments show that on typical scenarios time-parallel simulation leads to a significant speedup while maintaining a high level of accuracy.

1 Introduction

The development of communication systems requires rigorous performance studies. Analytical performance studies can only be carried for simple models of complex systems; such studies tend to provide a basic understanding of the system behavior and qualitative results. Simulation, on the other hand, is often used for quantitative performance study of more realistic models of complex systems. Wireless ad hoc networks are rarely amenable to analytical performance analysis due to the complexity of the models involved. Simulation has emerged as an important tool for the study of wireless ad hoc networks, but it is commonly used to study networks with a few hundred nodes for a relatively short period of time.

Nowadays systems consisting of a few thousand nodes are rather common. Indeed, let us consider a simple scenario: there are 120 students in a classroom; each student has a cell phone (GSM source), a PDA, and laptop (two 802.11b WiFi sources). There are five Bluetooth sources: PDA, laptop, cell phone, headset, and mouse. Some of the students might have WiFi enabled cameras, Bluetooth enabled audio players, and matching head phones. All in all, it does not seem out of the ordinary to have 3 WiFi and 7 Bluetooth sources per person. Thus, even without considering that many of the WiFi nodes have a transmission range long enough to cover neighboring classrooms as well, in order to study the networking environment of one classroom we have to simulate a system with 1200 wireless sources operating in the same frequency band.

A wireless network with $1,200$ nodes pushes the limits of serial simulators such as ns-2. Even when feasible, such simulations require a significant amount of computing power and can take a very long time. An alternative is to resort to space-parallel or time-parallel simulation. In many instances, e.g., in the classroom environment, every transmission can be received by, or at least interfere with the transmissions of many if not all the other nodes; the fully connected dependency defeats the purpose of spatial partitioning.

In this paper we present an approach for time-parallel simulation of wireless ad hoc networks. Time-parallel simulation has been used for some time; its applicability is strongly dependent of the simulated phenomena and can yield exact results only for systems whose behavior is modeled by regenerative stochastic processes. However, as we will show, good approximations can be obtained quickly, and the quality of the approximation can be improved through iteration.

Our time-parallel simulation of wireless ad hoc networks uses a traditional simulation package coupled with new techniques for the practical implementation of the simulation. We develop a methodology to analyze the accuracy of the simulation and introduce the notion of perturbation of measurements; then we present a layer-by-layer analysis of the impact of the perturbations on the performance of the network. This approach allows us to predict the accuracy of the simulation results through an analysis of the protocols involved at each layer.

This paper is organized as follows. We survey related work in Section 2, then we present the basic concepts for time-parallel simulation of wireless networks in Section 3. We also introduce a model of the propagation of perturbations in wireless networks, the impact of the protocols at the various layers of the networking stack. We apply these considerations to propose several methods for improving the speed and accuracy of time-parallel simulation. Section 4 presents the implementation of the theoretical models developed in the previous section with the widely used ns-2 simulator. We find that some of the theoretical concepts can be immediately incorporated in simulation studies conducted with existing simulators such as ns-2 while others are very difficult due to the particularities of the implementation. A series of experiments investigating the speedup and precision of the proposed method for typical wireless network simulation scenarios are presented in Section 5. We summarize our results in Section 6.

2 Related work

Parallel discrete event simulation (PDES) reduces the overall execution time by parallel execution of the simulation on multiple processors. There are two main avenues for parallel simulation: space-parallel simulation (distributed simulation), and time-parallel simulation. In the space-parallel simulation approach [17, 27], the simulation model is decomposed into a number of components on a spatial basis. Each component is modeled by a *logical processor*. Logical processors establish a communication mechanism among each other to avoid or fix possible causality errors. There are two general mechanisms to avoid/correct causality errors: optimistic mechanisms and conservative mechanisms. With optimistic mechanisms [22], a processor can execute an event e without the knowledge of its prior events, and state recovery methods [34, 35, 37] are required to restore the state of the simulation once causality errors are detected. Instances of optimistic space-parallel simulations include [9, 11, 12, 16, 35, 43]. In conservative simulations, a processor does not execute an event e until all events that may affect event e are executed. Instances of conservative space-parallel simulations include [19, 25, 28, 45].

Load balancing in PDES refers to distributing the workload over different processors evenly. An efficient load balancing scheme can greatly improve the speedup of PDES, due to the fact that the overall progress of the simulation is decided by the progress of the slowest processor. Various load balancing approaches for space-parallel simulations can be found in [2, 4, 8, 13–15, 18, 36, 38, 44].

The Parallel/Distributed Network Simulator (PDNS) [45] project uses a space-parallel simulation approach based on the ns-2 network simulator [39]. However, the applicability of PDNS is limited to *wired* networks, and the traffic simulated at different spatial partitions cannot affect each other.

The SWiMNet parallel simulator [5–7], is used for the simulation of personal communication services (PCS) networks with fixed channel assignment by specifying fine grained mobility, variable call process, and arbitrary coverage area. It is based on a combination of optimistic and conservative paradigms and makes use of the event precomputation by the model independence within the PCS model.

WiPPET [31], an optimistic parallel simulator for evaluating the performance of wireless protocols, exploits the parallelism of multi-channel radio networks either by geographic locations of the resources or by different radio channels.

Table 1 presents a comparison of PDNS, SWiMNet, WiPPET and the time parallel simulation approach proposed in this paper.

Table 1. A comparison of PDNS, SWiMNet, WiPPET and the time-parallel simulation approach (TPS) proposed in this paper

	PDNS	**SWiMNet**	**WiPPET**	**TPS**
Application domain	wired networks	PCS networks	multi-channel radio networks	wireless ad hoc networks
Parallelism	space-parallel	space-parallel	space-parallel	time-parallel
Error Control Scheme	conservative	conservative / optimistic	optimistic	state matching
Main Issues	The interference between different logic processors must be limited. The optimistic approach requires a large amount of memory.			Efficient estimation of the initial state of the simulation segments.

In the time-parallel simulation approach [1, 20, 21, 26, 40, 41], the long simulation interval is partitioned into smaller adjacent simulation intervals, and each simulation interval is assigned to a processor with a *guessed* initial state. The simulation terminates when the final state of each interval matches the initial state of its successive interval. Thus, *state matching* is one of the key problems of time-parallel simulation. In [26], the authors propose a time-parallel simulation algorithm based on state matching. A simulation is defined as *partial regenerative* if there exists a subset of the system state variables such that the subsystem represented by the subset can repeat its state infinitely many times. The system is then partitioned at the *regeneration points* which mark a *regenerative sub-state*. In some cases the *regeneration points* of a regenerative simulation can be found without performing a detailed simulation; the state matching problem can be solved by performing a pre-computation [29, 30]. Wang and Abrams propose a *pre-simulation* to identify regenerative points based upon Markovian modeling [41].

Kiesling [24] mentioned that the widespread use of time parallel simulation is restricted by the state-match problem, due to the difficulty in identifying regeneration points, especially for models with complex states. Unfortunately, time-parallel simulation of wireless networks belongs to this category. Kiesling pointed out that the use of approximate solutions can facilitate the temporal decomposition of simulation models. Since errors are introduced, an error control method must be provided for approximate time-parallel simulation. The simulation starts with guessed incorrect initial states, and fix-up computations are re-executed to reduce the error. Although time-parallel simulation rarely allows us to obtain accurate results, approximate results [21, 23, 24, 40] can be produced efficiently. The initial version of the time-parallel simulation of wireless ad hoc networks is presented in [3].

3 Time-parallel simulation of ad hoc networks

3.1 The one-step time-parallel model

A simulation S of an ad hoc network is specified by a quadruple $(\mathcal{E}, A, \tau, \mathcal{I})$ where $\mathcal{E} = \{e_1, e_2, \ldots\}$ is a set of *planned events*, A is the geographic area, $\tau = [\tau^s, \tau^e)$ is a time interval and $\mathcal{I}$ is the initial state of the network at time τ^s. The output of the simulation is the simulation trace $\mathcal{T} = \{t_1, t_2, \ldots\}$ and the final state $\mathcal{F}$ at time τ^e. The simulation trace is a series of events which includes the planned events ($\mathcal{E} \subset \mathcal{T}$) as well as events which are consequences of the planned events.

The idea behind time-parallel simulation is to replace the simulation S with a number of smaller *simulation segments*, which operate on the full area A, but on subsets of the time interval τ. First, we describe the most straightforward way for implementing time-parallel simulation through disjoint time intervals of equal length. In the following sections, we consider modifications to this basic model in order to achieve higher accuracy and/or speedup. In this basic model, we partition the time interval into m disjoint intervals of length $\tau_d = \frac{\tau^e - \tau^s}{m}$:

$$\tau_i = [\tau_i^s, \tau_i^e) = [(i-1) \cdot \tau_d, i \cdot \tau_d), \;\; i \in \{1, .., m\} \tag{1}$$

This also implies the partitioning of the planned event set:

$$\mathcal{E}_i = \{e_i \mid e_i \in \mathcal{E}, \;\; \tau_i^s \leq time(e_i) < \tau_i^e\} \tag{2}$$

The segments of the time-parallel simulation are specified by the quadruples $(\mathcal{E}_i, A, \tau_i, \mathcal{I}_i)$. As the segments are m times shorter than in the original simulation, their execution will take roughly m times shorter time, and they can be executed in parallel on m independent processors. The simulation trace of the full simulation will be the union of the simulation traces $\mathcal{T} = \mathcal{T}_1 \bigcup \ldots \bigcup \mathcal{T}_m$, while the final state of the network will be $\mathcal{F} = \mathcal{F}_m$.

The main difficulty pertains the calculation of the initial state $\mathcal{I}_i$. For the first segment, we have $\mathcal{I}_1 = \mathcal{I}$. For the other segments, the correct initial state is the final state of the previous segment $\mathcal{I}_i = \mathcal{F}_{i-1}$. However, using the final state of the previous simulation segment would require the in-order simulation of the segments, preventing any parallelism.

One crude approximation would be to assume that every simulation segment starts with the original state of the network $\mathcal{I}$. In this *one-step time-parallel simulation* the simulation segments are specified by $(\mathcal{E}_i, A, \tau_i, \mathcal{I})$.

Experiments show that the accuracy of the one-step time-parallel simulation is not sufficient for the practical purposes of wireless ad hoc network simulation. In the following, we analyze the source of errors in the time-parallel simulation of wireless ad hoc networks, estimate the magnitude of these errors, and propose methods to improve the accuracy of the simulation.

3.2 Measurements on the simulation trace

A simulation trace includes *planned events*, $(\mathcal{E} \subset \mathcal{T})$, as well as *new events* triggered by the planned events and calculated by the simulation process. For example the event corresponding to sending a packet may trigger a new event indication a collision with with another transmission. Every event e_i has a set of *properties*, e.g, the time of the event $time(e_i)$, the type of the event, the packet size, etc. Such information allows us to determine important measures of performance such as packet delay, packet loss ratio, throughput, and so on.

The purpose of running a simulation is to collect the information associated to the events in the trace. However, we are rarely interested in individual events, we are concerned with observations, or measurements, spanning longer time intervals, or the entire trace.

Measurements can be:

(a) *instantaneous* $M^i(t_c)$, reflecting the situation at the current time t_c.

(b) *cumulative* $M^c(t_c)$, reflecting the evolution of the measurement from the beginning of the simulation to the current time t_c. For every instantaneous measurement M^i, there is an associated cumulative measurement M^c:

$$M^c(t_c) = \int_0^{t_c} M^i(t)dt \tag{3}$$

(c) *average* $M^a(t_c)$. For every instantaneous measurement M^i, there is an associated average measurement M^a defined by:

$$M^a(t_c) = \frac{\int_0^{t_c} M^i(t)dt}{t_c} \tag{4}$$

An example is of instantaneous measurement is the *current transmission rate*, with the associated cumulative measurement *total transmitted data*, and the associated average measurement *average transmission rate*.

Many average measurements are expressed as *ratios* of measured quantities. For some of these measurements, we find it useful to define the complementary measure $C(M^a) = 1 - M^a$. For instance the ratio measure *average packet loss ratio* has its complement *average packet delivery ratio*.

3.3 Perturbations on a measurement

In the following, we investigate how an event *perturbs* the measurements of a simulation of wireless ad hoc networks. Our model accurately captures the underlying phenomena and provides a framework in which observations can be analyzed and predictions can be made.

We call an *approximate simulation for the measurement* M, related to simulation S, a simulation S' which generates a trace $\mathcal{T}'$ such that $M(\mathcal{T}') \approx M(\mathcal{T})$.

The *absolute error of simulation* for measurement M is

$$\delta_M = |M(\mathcal{T}) - M(\mathcal{T}')| \tag{5}$$

The *relative error of simulation* for measurement M is

$$\epsilon_M = \frac{|M(\mathcal{T}) - M(\mathcal{T}')|}{|M(\mathcal{T})|} \tag{6}$$

We define the relative accuracy for measurement M as $1 - \epsilon_M$. In most cases, the relative accuracy, expressed in percentages, is the most intuitive measure of the quality of an approximate simulation. For instance, a statement such as “the simulation has an accuracy of 98%” can conveniently express the fact that the relative error is smaller than 2%. The relative accuracy is an non-dimensional quantity, allowing us to compare the accuracy of different measurements.

However, the relative error suffers from the phenomena of error amplification for measurements which represent the average ratios for rare events. Let us consider a simulation scenario where out of 1000 packets sent 2 are lost; then we say that the packet loss ratio is $M_{\text{PLR}} = 0.002$. If we have an approximate simulation which finds only one lost packet, the relative error will be:

$$\epsilon_{\text{PLR}} = \frac{|M'_{\text{PLR}} - M_{\text{PLR}}|}{|M_{\text{PLR}}|} = \frac{|0.002 - 0.001|}{|0.002|} = 50\% \tag{7}$$

However, repeating the same calculation for the complementary measure packet delivery ratio $M_{\text{PDR}} = C(M_{\text{PLR}}) = 1 - M_{\text{PLR}}$:

$$\epsilon_{\text{PDR}} = \frac{|M'_{\text{PDR}} - M_{\text{PDR}}|}{|M_{\text{PDR}}|} = \frac{|0.998 - 0.999|}{|0.998|} \approx 0.1\% \tag{8}$$

Thus, the simulation appears very inaccurate for one measurement and very accurate for the complementary measurement. The physical phenomena behind this phenomenon is that the appearance of a rare event, for instance of a packet loss in a lightly loaded network is dependent on the coincidence of a number of factors. An approximate simulation might approximate well the probability of occurrence of those factors, but not the exact location and time where the event will take place. Therefore, the absolute error might be a better indicator of the accuracy of simulation for the measurements on rare events. The absolute error has the additional advantage that the absolute error of the measurement and the complementary measurement is identical:

$$\delta_M = |M(\mathcal{T}) - M(\mathcal{T}')| = |(1 - M(\mathcal{T})) - (1 - M(\mathcal{T}'))| = \delta_{C(M)} \tag{9}$$

Now we introduce the notion of a *perturbing event* e_p. We consider a simulation with the planned events $\mathcal{E} = \{e_1 \ldots e_n\}$ and a perturbed set $\mathcal{E}' = \mathcal{E} \bigcup e_p$. The perturbed set of planned events will lead to a simulation trace $\mathcal{T}'$ and, obviously, to perturbed measurements $M'(t)$. We are interested in the size and the temporal extent of the perturbations $\Delta M(t) = M'(t) - M(t)$. We say that:

– The perturbation has *no effect* on measurement M if $\Delta M(t) = 0, \ \forall t$.
– The event e_p creates a *time limited perturbation* in the measurement M which lasts until the *extinction time* t_e if $\Delta M(t) = 0, \ \forall t > t_e$.
– The event e_p has a *shift effect perturbation* on the measurement M if $\lim_{t\to\infty}(\Delta M(t)) = c$, with c being the *shift constant*. If there is a time point t_s with the property that $\Delta M(t) = c, \ \forall t > t_s$, we call the t_s *stabilization point*.

Property 1. If an event e_p causes a time-limited perturbation on the instantaneous measurement M^i with extinguishing time t_e, it creates a shift effect perturbation on the corresponding cumulative measurement M^c, with the stabilization point $t_s = t_e$. The shift constant can be expressed as:

$$c = \int_{t_p}^{t_e} \Delta M^i(t)dt \tag{10}$$

– We say that an event causes a *destabilizing perturbation* of measurement M if $\nexists \, t_s > t_p$ such that $\Delta M = c, \ \forall t > t_s$.

3.4 Propagation of perturbations in wireless ad hoc networks

As a first approximation, the events in a wireless ad hoc networks are caused by individual transmissions of packets. We consider the perturbing event to be the insertion of a new packet in the network. The removal of the packet is not considered separately, because it is equivalent to simply reversing the perturbed and the original system. As we are concerned about the absolute values of the difference on the measurements of these systems, the reversed system will yield the same conclusions. An additional type of perturbation we consider is the perturbation of the initial state – that is, the simulation starts with a different initial state than expected. We discuss this type of perturbation separately.

For events representing the addition or removal of a single packet, the immediate affect of the perturbations is usually minimal. However, the networking protocols deployed at various levels of the networking stack can amplify or (in some cases) reduce the impact of the perturbations. In the following, we investigate the impact of the protocols deployed at the various layers of the networking stack on the perturbation produced by the insertion of a single packet.

Physical layer An inserted packet will perturb the physical layer measurements only for the duration of the packet transmission. The time it takes to transmit a packet can be calculated as follows. Assume that we send a packet of 1536 bit, the maximum length supported by the 802.11b protocol. The transmission rate of 802.11b is 1.375 Mbps. The time required to send this packet is composed of:

– Distributed Inter-Frame Space (DIFS), set to 50μs
– Data packet transmission: 192μs for the preamble + $1536/\ 1.375Mbps = 192\mu s + 1118\mu s$
– Small Inter-Frame Space (SIFS), set to 10μs
– 802.11 ACK packet: $192\mu s + 14/1.375Mbps = 203\mu s$

Thus, the total time becomes $2084\mu s$, or approximately, 2 ms. This might change slightly for different protocols, but the order of magnitude remains the same. We conclude that the perturbation in the physical layer due to a new packet is time-limited, with a very short (2 ms) extinction time.

MAC layer The perturbation caused by an inserted packet in the MAC layer depends on the load of the network, and the type of the inserted packet. If the packet is a control packet such as ACK, RTS, or CTS, its influence extends beyond the time frame covered. For instance, receiving an RTS packet in a CSMA-CA network forces the node to refrain from transmitting for the duration specified in the packet. The size of this interval can be as long as the maximum packet transmission time, on the order of magnitude of 2μs.

Delaying the sending of a packet can in its turn delay the sending of the response or follow-up packets, creating a ripple effect. Normally, this will appear as a time-limited perturbation. For a TDMA (Time Division Multiple Access) type protocol, for a channel with the capacity of n bps, with a load of $\nu \leq 1$, and a packet size of l, the extinction time can be estimated as:

$$t_e = \frac{l}{n \cdot (1 - \nu)}$$

Note that for a full channel, $\nu = 1$, the perturbation becomes a shift effect perturbation.

For CSMA-CD (Carrier Sense Multiple Access with Collision Detection) protocols, the perturbation will extend to the length of the contention window. 802.11b uses an exponential back-off algorithm where the contention window can vary between CVmin and CVmax. Typical values of CVmin are between 7-15 and of CVmax are between 7-255 with the numbers being multiples of a slot time which is 20μs in 802.11b. Thus, the influence of initial collision can extend to 5000μs. However, if the channel is very busy, the initial collision can lead to further collisions down the line. Furthermore, any collision is extending the collision window through exponential back-off, which will be only gradually reduced.

In conclusion, on highly loaded networks, the insertion of a packet creating a collision can cause perturbations of up to several seconds in the worst case.

Routing layer Perturbations triggered by packet insertion at the routing layer are heavily dependent on the nature of the protocol and the inserted packet. The critical question is whether the packet will change the routing of future packets or not. If the packet is part of an established flow of application layer data, it will most likely not affect the routing of the other packets. If the packet is the first packet of a new flow, and the routing protocol is a reactive one, the packet will establish a new route. This frequently requires broadcast of routing information, which in its turn triggers the transmission of additional packets. Although this appears to be a destabilizing perturbation, the extent of this flooding is carefully constrained by the routing protocol and it will last at most several seconds.

Proactive routing algorithms deploy a routing table periodically updated by routing packets. The insertion of a routing packet triggers a perturbation in the network by changing the routing table of a node, and this will affect the routing of future packets. This is a major perturbation, affecting a large number of packets and a time interval on the order of minutes. A routing table perturbation is extinct when the routing tables of the original and the perturbed system re-converge. This situation occurs when:

– The modified routes are superseded by new, independently discovered routes, in both the original and the perturbed system (for instance, as a result of node mobility).
– The original system acquires the same routes as the perturbed one.
– The modified routes expire through a timeout and the routing table returns to the unperturbed version.
– An external command or a predetermined timeout flushes partially or completely the routing tables, forcing the recomputing of all routes.

Although it is technically possible to imagine a routing protocol where the loss or addition of a single routing packet would change the routes indefinitely, virtually all protocol designers, in their quest to make the protocols more reliable, have adapted features which make the routing tables converge; this has the indirect effect of limiting the perturbations and improve the accuracy of time-parallel simulations.

Transport layer and application layer Finally, we investigate how a perturbing event affects the transport layer and the application layer protocols. The major difference here is between reliable or non-reliable protocols. Let us consider the case of the most frequently used reliable transport protocol, TCP. The loss of a single TCP packet can significantly perturb the subsequent TCP flow: the packet will be retransmitted, the transmission window reset to its minimal value, which will then extend through the slow start algorithm. Thus, the loss of a single packet can exert an influence over the network for several tens of seconds.

In the case of the UDP protocol, which does not implement reliable transmission, the perturbation is minimal or nonexistent at the transport layer. However, applications which deploy UDP at the transport layer frequently use application layer protocols to control the flow of data. For instance, the multimedia streaming application RealPlayer is using an application level byte stream protocol RTP (Real-time Transport Protocol, specified in RFC 1889) for the transfer of multimedia information, with UDP being the transport protocol. Packet losses are handled by a complex logic and actions involving the RTSP (Real Time Streaming Protocol), RTCP (Real Time Control Protocol), SDP (Session Description Protocol) and, of course RTP.

We conclude that perturbations have effects up to the application layer. However, at the application layer, the behavior of the system becomes very complex, and in some cases, small perturbations can have major effects on the state of the system. This problem is not specific to time-parallel simulation. Most simulation studies of wireless ad hoc networks consider scenarios with constant or variable bit rate generic UDP sources, without application level protocols deployed in the simulator. When application level flow is simulated, it is done by an artificially generated stream which replicates the properties of a packet stream generated by the application level protocol, without actually deploying the application in the simulator [42]. In these types of models perturbations propagate only over very short time frames, thus creating an environment favorable for time-parallel simulation.

Until now, we have considered only perturbations caused by a single event. In the following, we discuss the perturbations caused by the interaction among two or more independent events:

Dual events. Dual events cancel each other; their successive occurrence does not change the state of the system. For instance, a reboot sequence is represented by two events: a node failure, followed after a short time by the node recovery. Individually, each event causes a perturbation, however their effects cancel each other. The state of the system will be the same as if none of the events occurred. This favorable case rarely occurs in practice.

Idempotent events. Idempotent events lead to the same state regardless if only one, a subset, or all of them occur. For instance, two subsequent missed routing table updates create the same effect an incorrect routing table entry, which requires the need to run a path discovery process. Idempotent events appear whenever some component of the system has a special "correct" state, while the other, "incorrect" states are functionally identical. Such system components are relatively frequent; examples are routing table entries or established end-to-end connections.

Independent events. Independent events lead to the same state of the system regardless of the order in which they occur; the overall perturbation of state caused by the events is additive. Their compound effect can be studied considering the perturbations caused by the individual events independently.

Mutually amplifying events. Such events lead to drastic alteration of state; the perturbation caused by the a sequence of such events is significantly larger than the perturbations caused by the individual events occurring independently. For example, an event causes a routing table to be overwritten with incorrect values and the second event propagates the incorrect routing table to all the nodes in the network, causing the network failure.

Naturally, mutually amplifying events are the worst case scenario from the point of view of the accuracy of the time parallel simulation. We need to carefully consider whether they can occur in the considered scenarios. Note that in our hypothetical example, the mutually amplifying events caused a total interruption of the communication in the simulation. Practically deployed networking stacks contain build-in guards against such types of unstable behavior. Mutually amplifying events are clearly possible during the experimental testing of new routing or MAC protocols. The observation of such an unstable behavior is an important feedback to the developers; at the same time, they need to be aware that the errors in the time-parallel simulation of such unstable systems are significantly higher.

We conclude that inter-event interactions yield perturbations which are identical or smaller than the perturbations considered individually, for most practically deployed, stable protocols. Mutually amplifying events, however, are possible if the deployed protocol stack shows unstable behavior (most frequently, as a result of experimental code). In these cases the accuracy of the simulation will be much lower.

3.5 Techniques for improving the accuracy of time-parallel simulations

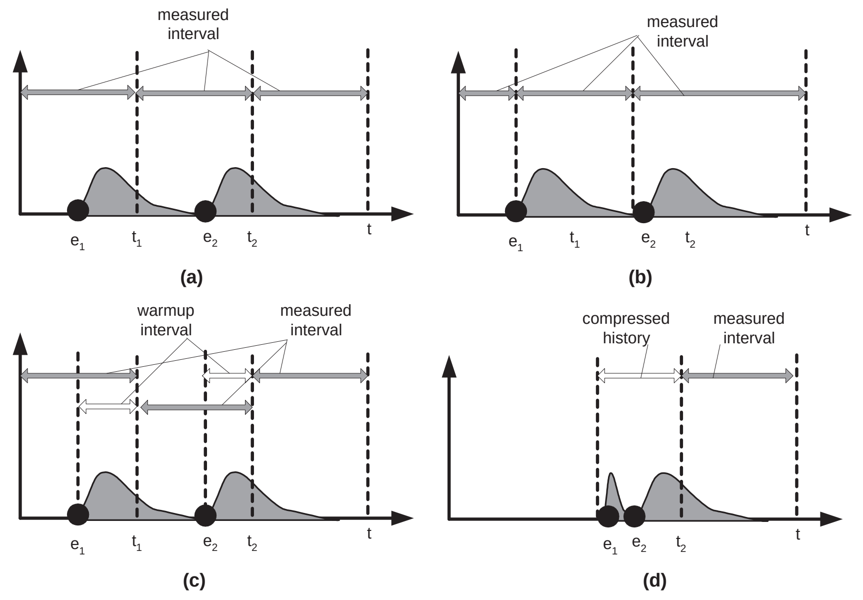


Fig. 1. An illustration of several techniques for improving the accuracy of time-parallel simulations. (a) One-step time-parallel simulation with segments of equal length. (b) Time interval shift. (c) Warmup intervals. (d) Warmup with history compression.

Let us now return to the one-step time-parallel simulation described in Section 3.1, and consider the sources of errors in the approximation. For instance, in Figure 1-a we see a partitioning of the simulation interval $[0, t]$ into three equal time intervals: $[0, t_1 = t/3]$, $[t_1 = t/3, t_2 = 2t/3]$ and $[t_2 = 2t/3, t]$. The event e_1, in the first segment is creating a perturbation which is not extinguished until the middle of the second segment. As the two segments are simulated independently, the processor simulating the second segment does not know about the perturbation which leads to an inaccurate simulation. Let us now review several techniques through which this inaccuracy can be reduced or eliminated.

Time interval shift. One way to increase the accuracy of the simulation is to select the boundaries of the simulation intervals such that the perturbations are completely contained in the segments, as shown in Figure 1-b. These points are the equivalent of the regenerative states of stochastic processes. If we can not find states where all the perturbations are extinguished, we can search for points where there are a comparatively smaller number of ongoing perturbations (partial regenerative states).

There are several problems with this approach. First, there might not be any regenerative states in the simulation, or their distribution might be such that it does not lead to segments of size appropriate for parallelization. The most difficult problem, however, is finding regenerative or partially regenerative states in a simulation without first running the simulation itself. There are certain circumstances when the identification of such points is possible. If a routing protocol periodically flushes the complete set of routing information, that instance corresponds to a regenerative state. Similarly, long periods of silence can be used as regenerative states, and they can be identified with an initial analysis of the scenario.

Warmup interval. This technique relies on separating the timespan of the simulation segment into the *warmup interval* followed by the *measurement interval*. During the warmup interval, we perform the simulation but do not record any measurements. Thus, the simulation performed during the measurement interval will be more accurate, because it will consider not only the events occurring during the measurement interval, but also the perturbations caused by events which occurred during the warmup.

The ideal size of the warmup interval is the shortest period which contains all the perturbing events which generate perturbations extending into the measurement interval (see Figure 1-c). The size of the required warmup can vary between various segments, the first segment does not require a warmup. Unfortunately, for practical cases we can not accurately compute the ideal length of the warmup interval without first running the simulation.

Warmup with compressed history. In a practical run of time-parallel simulation with warmup, the size of the warmup period can be significantly longer than the measured interval; thus most of the computation time is spent into the simulation of the warmup interval, which does not contribute to the measurements. Traditionally, the warmup interval is the exact copy of the simulation scenario for a period before the measured interval. We can replace the warmup interval with a shorter and/or simpler simulation interval which, however, would yield the same results. For instance, we can remove all the events from the warmup period which do not produce perturbations at the measured point. For the events which produce perturbations, we might be able to replace them with events easier and faster to simulate. We call this modified warmup interval a *compressed history* (see Figure 1-d).

Initial state approximation (ISA). In this technique, we are computing an approximation of the initial state of the segments without previous execution of the simulation. We had seen that for a typical simulation of a wireless ad hoc network the perturbation with the largest impact on the measurements relates to changes in the routing table. As most simulations start with an empty routing table which will be filled in during the initial phase of the simulation, this is a major source of errors. Approximating the routing table at the beginning of each simulated segment can thus significantly increase the accuracy of the simulation.

Composite methods. The methods of improving the precision of the time-parallel simulation can be used in conjunction with each other. The warmup can be composed of two periods: one of compressed history (for instance, covering the complete time-frame from the start of the simulation), followed by a warmup period operating on the unmodified planned event list. The initial state approximation method can be used to approximate the state at the beginning of the warmup period. Time period shift can be deployed in conjecture with any of the methods.

4 Implementation of time-parallel simulation for ad hoc networks

In the following, we present our implementation of the time-parallel simulation using the ns-2 simulator [39]. The first method, based on the iteratively extended warmup of the simulation segments, can be deployed without changes to the ns-2 implementation of the protocols. An alternative method combines iteratively extended warmup and initial state approximation and produces better results; this method requires some knowledge of the implementation of the routing protocol and additional code to allow the initialization of the routing tables.

Let us now highlight the relevant features of the ns-2 simulator. We defined a simulation as the transformation the quadruplet $(\mathcal{E}, A, \tau, \mathcal{I})$ into the output consisting of the final state $\mathcal{F}$ and the simulation trace $\mathcal{T}$. While ns-2, is compatible to this definition, there are differences in our ability to extract, interpret and modify the different components.

The ns-2 trace files, $\mathcal{E}$ and $\mathcal{F}$ are very easy to create, parse and filter, they consist of lines of text, one line per event. Once the simulation passes a given timeline, the output is immediately available, thus data can be extracted while the simulation is in progress.

In contrast, it is relatively difficult to generate and extract the initial and final state of the simulation. The data associated with the state is distributed across the protocol implementations; there is no standard format and no standard API through which this information can be accessed. While there is no obstacle in principle to access this information, this task needs to be handled from case to case, and it requires extensive knowledge of the implementation details of the protocols involved.

4.1 Time-parallel simulation with iteratively extended warmup

Our first method for implementing time-parallel simulation relies on iterative extension of the warmup interval. The approach allows us to obtain simulation results with increasingly higher accuracy *during the runtime of the simulation*. The simulation can be stopped when the desired accuracy is reached. This approach does not require the manipulation of the initial or the final state of the simulation, thus it is independent of the simulated protocols. Nevertheless, different protocols might converge at a different speed to the correct solution.

To implement the approach, we first decide on the number of simulation threads, m and choose m equally spaced starting points $(i-1)\tau/m$, where $i \in \{1,..,m\}$. The m threads of the simulation are started independently and they are defined by the quadruplets $(\mathcal{E}_i, A, [i\tau/m, \tau), \mathcal{I})$ respectively. The planned event set $\mathcal{E}_i$ of simulation thread i is defined as:

$$\mathcal{E}_i = \{e \mid e \in \mathcal{E},\ time(e) \geq \frac{i\tau}{m}\} \tag{11}$$

Note that each simulation thread starts with the same, normally empty, initial state $\mathcal{I}$. The first thread, started at τ_1=0 is performing a traditional, linear simulation. An example of this process, for the case of $m = 10$, τ=200s is illustrated in Figure 2.

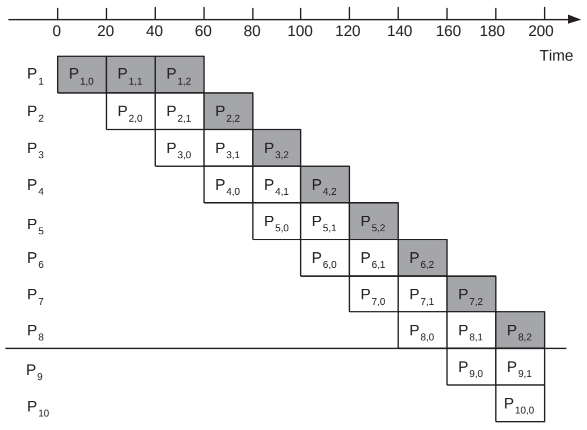


Fig. 2. Time-parallel simulation with iteratively extended warmup interval. The simulation progressed $3 \times 200/10 = 60$ seconds in each thread. The shaded rectangles show the current best approximations for the corresponding time segments.

We can obtain the first approximate solution starting from the moment when all the simulation threads have progressed at least τ/m in their simulation process. As the simulation progresses further, most time intervals were simulated by more than one thread, but as the threads started at different time points, the length of the warmup period with which the simulations were performed varies. The best approximation is obtained by stitching together the segments from the various simulation runs, *taking every segment from the simulation run where it was executed with the largest amount of warmup*.

For instance, the shaded rectangles in Figure 2 show the results obtained after each thread progressed over at least $3\tau/m$ simulation time. The time interval $[60, 80)$ is simulated by threads 2,3, and 4, the best approximation is provided by thread 2 with the longest warmup period.

4.2 Time-parallel algorithm with initial state approximation

In the previous section, we have seen how our implementation of iteratively extended warmup involves the threads of simulation specified by the quadruplets $(\mathcal{E}_i, A, [i\tau/m, \tau), \mathcal{I})$, that is, the simulation threads start with the same, usually empty, initial state $\mathcal{I}$. This is one of the major source of errors. To obtain an accurate simulation, we would need to start the simulation with an initial state $\mathcal{I}_i$ which describes the state of the simulation as would appear at the corresponding moment in a linear simulation run.

The initial state approximation method relies on running the simulation threads specified by $(\mathcal{E}_i, A, [i\tau/m, \tau), \mathcal{I}'_i)$ where $\mathcal{I}'_i \approx \mathcal{I}_i$ is an approximation of the initial state. The whole point of the method is that $\mathcal{I}'_i$ is computable without the need to run a serial simulation to the corresponding timepoint.

The analysis of Section 3.4 shows that the perturbations with the largest effect on the behavior of the network are the changes in the routing tables. Therefore, in our implementation of the initial state approximation, we will concentrate on the estimation of the state of the routing table at the simulated time point. Naturally, this method can be applied only to the routing protocols which use a routing table. The method can not be applied to purely reactive routing protocols such as AODV. On the other hand, reactive routing protocols have less state information, and approximating their initial state with an empty state yields less error. This conjecture is supported by our experiments in Section 5.2.

The approximation of the routing table needs to be implemented separately for every routing protocol, as both the routing table and its representation in the simulation code can vary between the different protocols. In the following, we describe our implementation for the DSDV routing protocol.

The implementation requires the understanding of the procedure used by the protocol to build its routing table. In DSDV, every node maintains a routing table which contains all available destinations, the next node to reach a given destination and the number of nodes necessary to reach that destination. In essence, this is a shortest path problem, complicated by the fact that (a) no global view of the network is available and (b) the mobility of the nodes can change the network connectivity and make the routing table obsolete. DSDV solves the first problem by deploying a distributed version of the Bellman-Ford algorithm. This algorithm relies on the iterative improvement of the routing tables through periodic or event-triggered broadcast of the routes to the immediate neighbors of the node. Even in a static network, it needs several send/receive cycles until the correct distance information is distributed all over the network. In a mobile network, the routing tables contain only an approximation of the correct shortest path information.

Let us now consider how we approximate the initial state of the routing table at time t. When running a simulation, the researcher is in the position of an omniscient external observer. Knowing the mobility models deployed in the simulation, one can obtain the correct location information of the nodes at any time point t_i. Using this information, an "ideal" routing table can be obtained by simply running Dijkstra's shortest path algorithm for each node. Note, that having a global view of the network, there is no point in running a distributed algorithm. The resulting routing table is the one the DSDV algorithm would converge to *if the nodes would be immobile* for a sufficiently long time; this table will be used as the approximate initial state $\mathcal{I}_i$ in the time-parallel simulation. In practice, the actual routing tables will be somewhat out of date due to node mobility.

This approach can be readily adapted to routing protocols which use a distance vector based routing table; routing protocols which use tables constructed based on different principles (e.g. geographic routing algorithms, directed diffusion, and so on) need different approximation methods.

5 Simulation Study

We present the results of an investigation of the performance, accuracy, and benefits of our methodology for time-parallel simulation. We first performed the simulation using the serial ns-2 simulator, obtaining a baseline result. Next, we repeated the simulation using the iteratively extended warmup model of time-parallel simulation as described in Section 4.1, with data concerning the throughput and packet loss ratio collected after every iteration. We repeated the simulations for the widely used Ad-hoc On-Demand Distance Vector Routing (AODV) and Destination Sequenced Distance Vector Routing (DSDV) protocols; for the table-driven DSDV protocol we run the simulation both with and without initial state approximation.

We are concerned only the *relative error* of the results; the absolute values of the measured quantities are not relevant for our study, they are dependent on the scenario and the deployed protocols, rather than the parallelization model. However, the considerations in Section 3.4 show that certain parameters of the scenario, such as the mobility of the nodes and the network load, affect the number and nature of perturbations. To quantify this influence we investigate the accuracy of the time-parallel simulation function of these scenario parameters.

In all our experiments, we found that the first iterations lead to a major decrease in the relative approximation error, followed by a smaller improvement in subsequent iterations. Thus, we find it useful to use a logarithmic scale for the presentation of the results.

5.1 Simulation setup

To run our simulations in a realistic setting, we chose a setting representative for practical scenarios. We consider a set of mobile nodes moving in a rectangular area, with a set of pre-determined communication patterns. The transmission range of the nodes is significantly smaller than the simulation area. The node-to-node communication is facilitated by a wireless ad hoc network routing protocol. One of the challenges of the scenario is that the mobility of the nodes changes the network topology and the routing of the packets.

Our scenarios use two well known protocols, representative of the major classes of wireless ad hoc routing protocols. DSDV [32] is a pro-active routing protocol in which the nodes maintain routing tables and perform actions to keep them up-to-date. AODV [33] is a reactive, on-demand routing protocol where the routes are established only as a result of explicit demand. These two classes of protocols exhibit different behavior in relation to the time-parallel simulation.

We use the "random waypoint" model [10] to simulate the node movement. Traffic patterns are generated by *Constant Bit Rate* (CBR) sources sending 512-byte UDP packets at a rate of 1 packet per second. The simulation area is 500×500 and the default number of nodes is 80. All the nodes have a transmission range of 100 meters. The scenario extends over a time interval of 600 seconds. Table 2 shows the default settings and the range of the parameters for our experiments.

Table 2. The default values and the range of the parameters for the simulation scenario

Parameter	**Default**	**Range**
simulation area	$500 \times 500(m^2)$	
number of nodes	80	
transmission range	100 (m)	
speed	1 (m/s)	1 - 21 (m/s)
pause time	15 (s)	
simulation time	600 (s)	
segment duration	30 (s)	10 - 60 (s)
number of CBR sources	20	4 - 40
CBR packet size	512 (bytes)	
CBR sending rate	4 (kbps)	

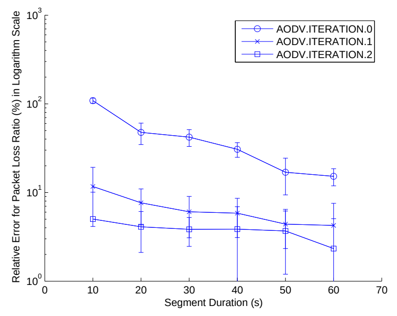
5.2 Achievable speedup

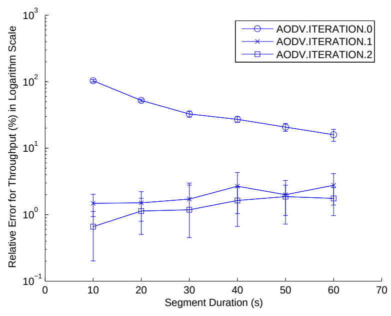
In our implementation m threads start at the simulation time $\tau_i = (i-1)\tau/m$ and run concurrently. The first approximate solution is obtained when all the threads have progressed at least to time $\tau_d = \tau/m$, thus the first approximation can be obtained m times faster compared to a serial simulation, ignoring the overhead required to extract and assemble the approximate trace file.

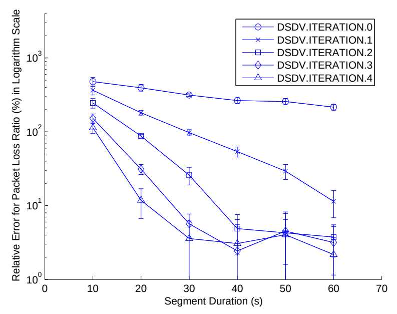
It would appear therefore that we want to increase m up to the number of available processors. However, the accuracy of the results decreases when m increases. Our approach is to run the simulation for several more iterations k, in each iteration the simulation progressing another τ/m on all threads.

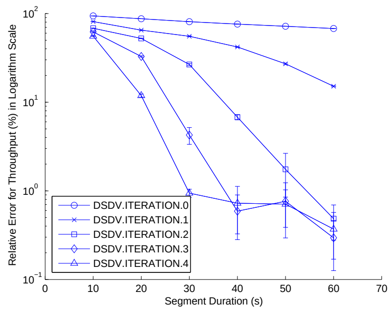
The iterations will stop when the desired accuracy is reached. For this paper, we assume that the desired accuracy is 95%. Let us assume that the simulation requires k_{95} iterations to reach this level of accuracy. The speedup of the time-parallel simulation will be $\eta = m/k_{95}$. The value of k_{95} depends on many factors: the choice of m, the length of the simulation, the deployed protocols and the simulation scenario. The use of initial state approximation can reduce the required number of iterations.

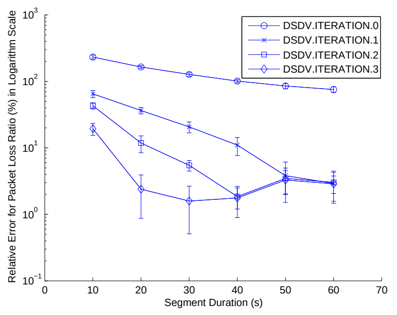
In a series of simulation experiments, we have measured the accuracy for various segment sizes τ_d = 10, 20, 30, 40, 50 and 60 seconds. In our experiments the simulated time being τ =600 seconds, this corresponds to values of $m =$ 60, 30, 20, 15, 12 and 10, respectively.


(a)


(b)


(c)


(d)


(e)

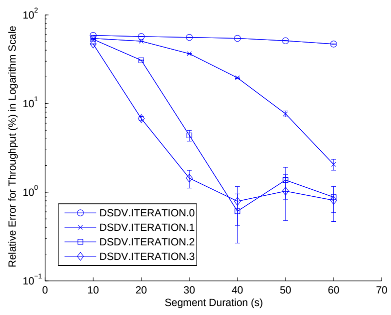

(f)

Fig. 3. The relative error of measurements in function of the segment duration at various iterations of the time-parallel simulations. The left side of the figure (diagrams a, c and e) show the measured packet loss ratio, the right side of the figure (diagrams b, d and f) the measured throughput. Diagrams a and b show the results for the AODV routing protocol, diagrams c and d for DSDV while diagrams e and f for DSDV with initial state approximation used in the time-parallel simulation.

We show the relative error for the packet loss ratio, Figure 3(left), and the relative error for the throughput, Figure 3(right). The lines marked *protocolname*.ITERATION.i show the relative error after the i-th iteration. The results show the average and the 95% confidence interval of 10 simulation runs.

The first observation is that for all simulations the accuracy in general is increasing with the number of iterations (although some accidental reversals are possible). As expected, for a given iteration, the accuracy is higher for longer segment sizes (that is, smaller m values). The accuracy of the throughput is in general higher than the packet loss ratio, thus we conjecture that the throughput is relatively less sensitive to perturbations than the packet loss ration.

There is a significant difference between the accuracy obtained at a given iteration for AODV (diagrams a and b) and DSDV (diagrams c and d). The proactive, table-driven DSDV protocol has a much lower accuracy for a given iteration, than the reactive AODV protocol. This is especially noticeable for the packet loss ratio. This validates our inference from Section 3.4. We note a significant increase in the accuracy for every iteration.

Table 3 shows a different view of this data, which might be more important for a researcher. With our assumption that the desired accuracy is 95%, the table shows for each m and corresponding τ_d value the number of iterations necessary to achieve that accuracy k_{95}, the actual accuracy obtained $(1 - \epsilon)$ and the overall speedup of the simulation η.

A conclusion to be drawn from Table 3, is that time-parallel simulation leads to a significant speedup. The speedup is 20 times for AODV, 4.28 times for DSDV without ISA, and 8.6 times for DSDV with ISA. In general, the speedup is greater for reactive routing protocols. Initial state approximation with the model described in Section 4.2 can significantly increase the speedup for proactive, table-driven protocols as well, but it comes with the disadvantage that it requires additional code besides the standard ns-2 protocol implementation.

Table 3. The number of iterations k_{95} needed, the accuracy $(1 - \epsilon)$ and the speedup η, as a function of segment duration, for the simulation of AODV, DSDV without ISA and DSDV with ISA

m - No. of threads	τ_d - Segment duration (s)	**AODV**			**DSDV w/o ISA**			**DSDV with ISA**		
		k_{95}	$1-\epsilon$	η	k_{95}	$1-\epsilon$	η	k_{95}	$1-\epsilon$	η
60	10	3	95.6%	20.0	14	97.3%	4.28	7	98.4%	8.6
30	20	2	95.4%	15.0	8	96.9%	3.75	4	97.6%	7.5
20	30	2	95.4%	10.0	5	96.7%	4	4	98.4%	5.0
15	40	2	95.8%	7.5	4	96.5%	3.75	3	98.1%	5.0
12	50	2	95.8%	6.0	4	97.7%	3	2	96.3%	6.0
10	60	2	95.7%	5.0	3	97.3%	3.3	2	97.2%	5.0

5.3 The influence of the scenario parameters on the accuracy

Our results show that a significant speedup can be achieved, while the accuracy is dependent on the choice of the routing protocols. In a series of experiments, we investigated the influence of other parameters of the scenario on the accuracy of the time-parallel simulation. In these experiments, we used a segment duration τ_d =30 seconds corresponding to m =20 simulation threads. For all the measurements, we measure the relative errors of packet loss ratio and throughput for AODV, DSDV without ISA, and DSDV with ISA.

Figure 4 shows the relative errors at various iterations function of the network load. The network load is simulated by varying the number of CBR sources (from 5 to 45). Diagram 4-a shows the relative error of the packet loss ratio and diagram 4-b shows the relative error for the throughput. We see a very slight tendency for the relative error on packet loss ratio to increase with the throughput for the early iterations. In general, however, the relative error remains almost constant across the range of the CBR sources considered. At the third iteration, the relative error was below 5% for all CBR values.

Figures 4-c and 4-d show the relative error for packet loss ratio and throughput, respectively, for the simulation of the DSDV protocol. Interestingly, the relative error for packet loss ratio shows a decreasing trend with the number of CBR sources for the early iterations and a slightly increasing trend for the later iterations. For the throughput, the relative error is almost constant for a given iteration, independently of the network load. These slight trends notwithstanding (some of which might be an artifact of the experimental setup), in general the relative error consistently decreases with the number of iterations for both packet loss ratio and throughput, for all the tested values of CBR sources.

Figures 4-e and 4-f show the values for DSDV with the initial state approximation approach being used in the time-parallel simulation. The trends are essentially the same as in the case without initial state approximation. However, for the equivalent iterations the relative error is significantly lower. For higher network loads, we see that the relative error for packet loss ratio is actually very close for iterations 2 and 3, and in one case, for 32 CBR sources, the iteration 3 shows a slight increase in the relative error over iteration 2 (although both errors are lower than 5%). This reversal indicates the limits of the improvement obtainable with the initial state approximation approach.

In conclusion, we find that the relative error of the packet loss ratio and throughput shows only very slight dependence on the network load. The general trend is that the relative error consistently decreases with the number of iterations.

Figure 5 shows the same set of measurements test function of the average mobility of the network nodes, ranging from 1 to 21 m/s. For most experiments, the results show that the relative error has a slight tendency to decrease when node mobility increases. A plausible explanation is that in a highly mobile network the routing tables and the cached flow entries are recomputed more frequently, thus limiting the influence of perturbations. Other than this, all the previously observed tendencies remain valid for all the possible values of node mobility. The relative error decreases with the number of iterations for both the packet loss ratio and the throughput. In general, AODV converges to a relative error of less than 5% in three iterations. DSDV converges much slower, but the convergence can be sped up using initial state approximation. Overall, the influence of the mobility on the accuracy is minor and predictable.

We conclude that the only significant parameter of the scenario is the choice of the routing protocol and, for pro-active routing protocols, whether initial state approximation was deployed or not. This is a favorable result, because it limits the number of variables a researcher needs to control for a time-parallel simulation.

6 Summary and Future Work

In this paper we described a methodology for time-parallel simulation of wireless ad hoc networks. We presented a quasi-formal analysis of perturbations, which gives us some understanding of the source of errors in one-shot time-parallel simulations. Based on a layer-by-layer analysis of the propagation of perturbations in the wireless networking stack, we proposed several avenues for improving the accuracy of the time-parallel simulation. Building on these considerations, we described an implementation of time-parallel simulation based on the ns-2 simulator. The techniques deployed are the iterative extension of the warmup period, and initial state approximation for the proactive, table-driven routing protocols. A series of experiments showed that a speedup between 5-20 times can be obtained for an accuracy of 95%, depending on the choice of the routing protocol and whether initial state approximation was deployed or not. In general, the simulation reactive routing protocols shows a higher accuracy and/or speedup than the one for proactive, table-driven protocols. We show, however, that the accuracy is relatively independent on other parameters of the scenario, such as network load or node mobility.

Time-parallel simulation can be used to study measures of performance such as packet loss ratio and throughput, but there are others measures such as end-to-end delay, that require a different approach.

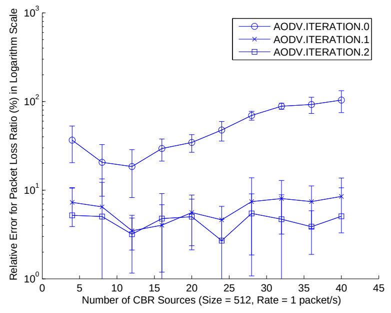
Future work include improved initial state approximation techniques that would bring the speedup of the table driven protocols closer to the one for reactive protocols. We are also considering more sophisticated models to predict the accuracy of the simulation for arbitrary scenarios and combinations of networking protocols. Finally, we plan to develop software components for time-parallel simulation of wireless ad hoc networks on cluster computers, without the requirement of in-depth knowledge of the protocol implementation.

Acknowledgment

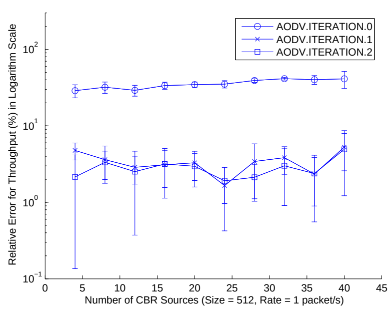
The research reported in this paper was partially supported by National Science Foundation grants ACI0296035 and EIA0296179.

References

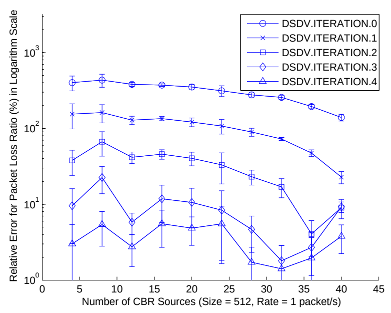
1. S. Andradóttir and T. J. Ott. Time segmentation parallel simulation of networks of queues with loss or communication blocking. *ACM Transactions on Modeling and Computer Simulation*, 5(4):269–305, 1995.



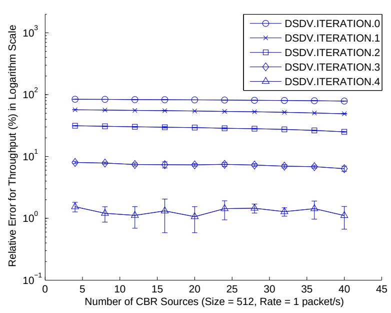
(a)



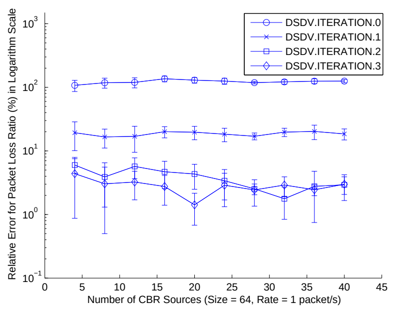
(b)



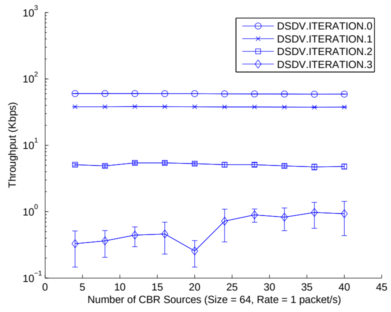
(c)



(d)

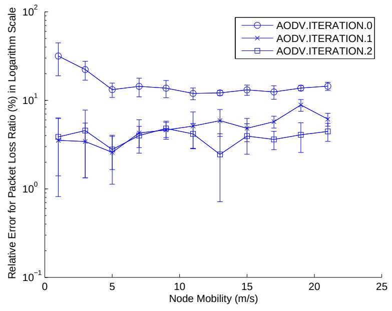


(e)

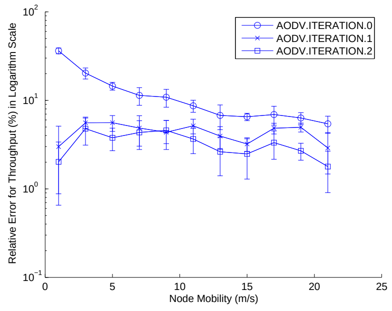


(f)

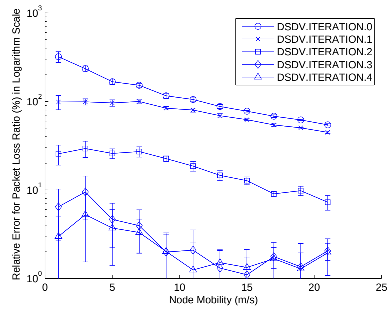
Fig. 4. The relative error of measurements in function of the network load at various iterations of the time-parallel simulations. The left side of the figure (diagrams a, c and e) show the measured packet loss ratio, the right side of the figure (diagrams b, d and f) the measured throughput. Diagrams a and b show the results for the AODV routing protocol, diagrams c and d for DSDV while diagrams e and f for DSDV with initial state approximation used in the time-parallel simulation.



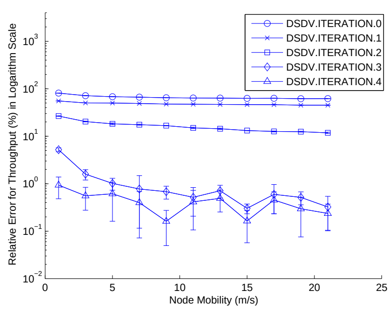
(a)



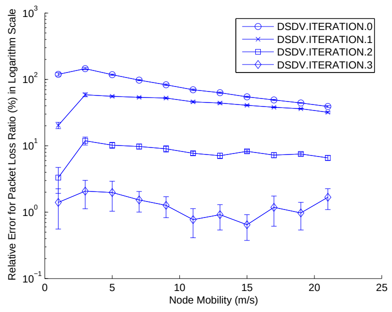
(b)



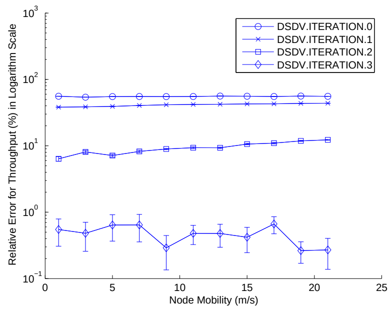
(c)



(d)



(e)



(f)

Fig. 5. The relative error of measurements in function of the node mobility at various iterations of the time-parallel simulations. The left side of the figure (diagrams a, c and e) show the measured packet loss ratio, the right side of the figure (diagrams b, d and f) the measured throughput. Diagrams a and b show the results for the AODV routing protocol, diagrams c and d for DSDV while diagrams e and f for DSDV with initial state approximation used in the time-parallel simulation.

2. H. Avril and C. Tropper. The dynamic load balancing of clustered time warp for logic simulation. In *PADS '96: Proceedings of the 10th workshop on Parallel and distributed simulation*, pages 20–27, 1996.

3. L. Boloni, D. Turgut, G. Wang, and D. Marinescu. Challenges and benefits of time-parallel simulation of wireless ad hoc networks. In *Proceedings of First International Conference on Performance Evaluation Methodologies and Tools (Valuetools-2006)*, October 2006.
4. A. Boukerche and S. K. Das. Dynamic load balancing strategies for conservative parallel simulations. In *PADS '97: Proceedings of the eleventh workshop on Parallel and distributed simulation*, pages 20–28, 1997.
5. A. Boukerche, S. K. Das, and A. Fabbri. SWiMNet: a scalable parallel simulation testbed for wireless and mobile networks. *Wireless Networks*, 7(5):467–486, 2001.
6. A. Boukerche, S. K. Das, A. Fabbri, and O. Yildiz. Exploiting model independence for PCS parallel simulation. In *Proceedings of 13th ACM/IEEE workshop on Parallel and Distributed Simulation*, pages 166–173, 1999.
7. A. Boukerche and A. Fabbri. Partitioning parallel simulation of wireless networks. In *Proceedings of 2000 Winter Simulation Conference*, pages 1449–1457, 2000.
8. A. Boukerche and C. Tropper. A static partitioning and mapping algorithm for conservative parallel simulations. *SIGSIM Simul. Dig.*, 24(1):164–172, 1994.
9. J. Briner. Fast parallel simulation of digital systems. In *Proceedings of Multiconference on Advances in Parallel and Distributed Simulation*, pages 71–77, 1991.
10. J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of MOBICOM*, pages 85–97, 1998.
11. C. D. Carothers, R. M. Fujimoto, P. England, and Y. B. Lin. Distributed simulation of large-scale PCS networks. In *Proceedings of the 2nd IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 2–6, 1994.
12. C. D. Carothers, R. M. Fujimoto, and Y. B. Lin. A case study in simulating PCS networks using Time Warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pages 87–94, 1995.
13. M. Choe and C. Tropper. On learning algorithms and balancing loads in time warp. In *PADS '99: Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 101–108, 1999.
14. E. Deelman and B. K. Szymanski. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. In *PADS '98: Proceedings of the twelfth workshop on Parallel and distributed simulation*, pages 46–53, 1998.
15. K. Devine, B. Hendrickson, E. Boman, M. S. John, and C. Vaughan. Design of dynamic load-balancing tools for parallel applications. In *ICS '00: Proceedings of the 14th international conference on Supercomputing*, pages 110–118, 2000.
16. R. M. Fujimoto. Time warp on a shared memory multiprocessor. *Transactions of the Society for Computer Simulation*, 6(3):211–239, 1989.
17. R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.
18. D. W. Glazer and C. Tropper. On process migration and load balancing in time warp. *IEEE Transactions on Parallel and Distributed Systems*, 4(3):318–327, 1993.
19. P. Heidelberger and D. Nicol. Conservative parallel simulation of continuous time Markov chains using uniformization. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):906–921, 1993.
20. M. Hoseyni-Nasab and S. Andradóttir. Parallel simulation by time segmentation: Methodology and applications. In *Proceedings of the 1996 Winter Simulation Conference*, pages 376–381, 1996.
21. M. Hoseyni-Nasab and S. Andradóttir. Time segmentation parallel simulation of tandem queues with manufacturing blocking. In *Proceedings of the 1998 Winter Simulation Conference*, pages 1487–1492, 1998.

22. D. R. Jefferson. Virtual time. *ACM Tranactions on Programming Languages and Systems*, 7(3):404–325, 1985.
23. T. Kiesling. Approximate time-parallel cache simulation. In *Proceedings of 2004 Winter Simulation Conference*, pages 345–354, 2004.
24. T. Kiesling. Using approximation with time-parallel simulation. *Simulation*, 81(4):255–266, April 2005.
25. Y. B. Lin and E. D. Lazowska. Exploiting lookahead in parallel simulation. *IEEE Transactions on Parallel and Distributed Systems*, 1(4):457–469, 1990.
26. Y.-B. Lin and E. D. Lazowska. A time-division algorithm for parallel simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(1):73–83, 1991.
27. V. K. Madisetti, J. C. Walrand, and D. G. Messerschmitt. Asynchronous algorithms for the parallel simulation of event-driven dynamical systems. *ACM Transactions on Modeling and Computer Simulation*, 1(3):244–274, 1991.
28. D. Nicol. Conservative parallel simulation of priority class queuing networks. *IEEE Transactions on Parallel and Distributed Systems*, 3(3):294–303, 1992.
29. I. Nikolaidis, R. Fujimoto, and C. A. Cooper. Parallel simulation of high-speed network multiplexers. *IEEE Conference on Decision and Control*, 3(1):2224–2229, 1993.
30. I. Nikolaidis, R. Fujimoto, and C. A. Cooper. Time-parallel simulation of cascaded statistical multiplexers. In *Proceedings of the 1994 ACM SIGMETRICS conference on measurement and modeling of computer systems*, pages 231–240, 1994.
31. J. Panchal, O. Kelly, J. Lai, N. Mandayam, A. T. Ogielski, and R. Yates. Wippet, a virtual testbed for parallel simulations of wireless networks. *SIGSIM Simul. Dig.*, 28(1):162–169, 1998.
32. C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of ACM SIGCOMM '94*, pages 234–244, 1994.
33. C. Perkins and E. Royer. Ad hoc On-demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 99–100, 1999.
34. F. Quaglia. A cost model for selecting checkpoint positions in time warp parallel simulation. *IEEE Transactions on Parallel and Distributed Systems*, 12(4):346–362, 2001.
35. F. Quaglia and A. Santoro. Nonblocking checkpointing for optimistic parallel simulation: Description and an implementation. *IEEE Transactions on Parallel and Distributed Systems*, 14(6):593–610, 2003.
36. P. L. Reiher and D. Jefferson. Dynamic load management in the time warp operating system. *Transactions of the Society for Computer Simulation*, 7(2):91–120, 1990.
37. H. M. Soliman and A. S. Elmaghraby. An analytical model for hybrid checkpointing in time warp distributed simulation. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):947–951, 1998.
38. T. K. Som and R. G. Sargent. Model structure and load balancing in optimistic parallel discrete event simulation. In *PADS '00: Proceedings of the 14th workshop on Parallel and distributed simulation*, pages 147–154, 2000.
39. VINT. The UCB/LBNL/VINT network simulator-ns (version 2). URL `http://www.isi.edu/nsnam/ns`.
40. J. J. Wang and M. Abrams. Approximate time-parallel simulation of queueing systems with losses. In *Proceedings of the 1992 Winter Simulation Conference*, pages 700–708, 1992.
41. J. J. Wang and M. Abrams. Determining initial states for time-parallel simulations. In *PADS '93: Proceedings of the seventh workshop on parallel and distributed simulation*, pages 19–26, 1993.
42. M. C. Weigle, P. Adurthi, F. Hernandez-Campos, K. Jeffay, and F. D. Smith. Tmix: a tool for generating realistic TCP application workloads in ns-2. *SIGCOMM Computer Communication Review*, 36(3):65–76, 2006.

43. F. Wieland. Practical parallel simulation applied to aviation control. In *Proc. of the 15th Workshop on Parallel and Distributed Simulation*, pages 109–116. ACM/IEEE Computer Society, 2001.
44. L. F. Wilson and D. M. Nicol. Experiments in automated load balancing. In *PADS '96: Proceedings of the 10th workshop on Parallel and distributed simulation*, pages 4–11, 1996.
45. PDNS – Parallel/Distributed NS. URL `http://www.cc.gatech.edu/computing/compass/pdns/`, 2004.