An Event-Synchronization Protocol for Parallel Simulation of Large-Scale Wireless Networks

Clinton Kelly, IV; Rajit Manohar
Computer Systems Laboratory
Cornell University
Ithaca NY 14853, U.S.A.

Abstract

We present a new conservative event-synchronization protocol, time-based synchronization*, for parallel discrete-event simulation of mobile ad hoc wireless networks. Simulators that use our protocol proceed at a scaled version of real time and send messages that correspond only to transmissions in the simulated network. We show that such simulators can maintain a constant execution time even as the sizes of the networks that they simulate grow. Moreover, we show that these simulators, when executed on a custom parallel architecture, are capable of simulating many networks* faster than real time.

1. Introduction

A *mobile ad hoc wireless network* (MANET) is a collection of mobile wireless nodes that form a temporary network without any infrastructure or centralized control. Researchers typically cite three uses for MANETs: emergency situations, military operations, and sensor networks [12, 17, 19, 20, 27]. MANETs used in such situations could conceivably contain several thousand nodes. Simulators that evaluate MANETs must therefore be capable of simulating large-scale networks. Researchers wishing to quickly simulate such networks typically use parallel discrete-event simulators (PDES) such as the Global Mobile Information System Simulator (Glomosim) [31], the Simulator for Wireless Ad Hoc Networks (SWAN) [19, 20], or QualNet [26]. These simulators all use *conservative event-synchronization protocols* to ensure that they produce the same results as a sequential simulator would.

In this paper, we present a new conservative event-synchronization protocol called *time-based synchronization* (TBS). We have designed this protocol such that a TBS-based simulator running on a custom, fine-grained message-passing multiprocessor called the Network on a Chip (NoC) can simulate extremely large MANETs faster than real time.

The rest of this paper is organized as follows. We begin by describing time-based synchronization (Section 2). We then describe how a typical discrete-event simulator models the behavior of a MANET with events and discuss how to change such a simulator to take full advantage of the benefits offered by TBS (Section 3). Next we describe the NoC and discuss why our simulator requires such specialized hardware (Section 4), and we show simulation results that demonstrate the improvement in simulation performance that a TBS-based simulator running on the NoC offers (Section 5). Finally, we discuss related work (Section 6) and summarize our research and our future plans (Section 7).

For the rest of this paper, we will consider a parallel discrete-event simulator to be composed of N logical processes, $LP_0, \ldots, LP_{N-1}$, which communicate by sending messages containing time-stamped events. Each logical process has the following components: (1) the state variables that correspond to the part of the simulated physical system that the LP represents, (2) a time-ordered event queue, and (3) a local clock whose value equals the timestamp of the LP's most-recently-executed event.

2. Time-Based Synchronization

An LP in a simulation using a conservative event-synchronization protocol must obey the local causality constraint [9]. If such an LP has an event with timestamp T at the head of its event queue, it cannot execute this event until it is *sure* that it will not later receive a message with a timestamp earlier than T.

In a simulation using the null-message protocol [3, 4], an LP receives messages via *incoming-message queues* (one for each LP that can send messages to the LP in question). Each such queue has a "clock," which equals the timestamp of the last message that the destination LP removed from the queue. Because the messages sent on each queue are guaranteed to have non-decreasing timestamps, an LP will become "sure" that it can execute the event at the head of its event queue when the clocks of all of its incoming-message queues are later than or equal to T. To ensure progress, LPs send timestamped null messages, which do not contain actual events, but instead contain an implicit promises that the senders will not send to the receivers any messages with timestamps earlier than the timestamps of the null messages.

An LP in a simulation using TBS becomes sure that it can execute the event at the head of its event queue when *the timestamp of the event is less than the the scaled version of the elapsed real time since the simulation began.* If the simulation has been executing for time t, then an LP can execute the event at the head of its event queue when

$$T < s \times t, \tag{1}$$

where s is the "time scale" of the simulation (for the rest of this paper, we will use t to represent the elapsed real time since a simulation began, and s to represent the time scale). When this inequality is true, we say that the event in question is *executable*. It is easy to see that a PDES using TBS will execute correctly as long as every event arrives at its destination LP before it is executable (see [14] for a proof). That is, an incoming message with timestamp T arriving at time t must satisfy

$$T \geq s \times t. \tag{2}$$

Note that, unlike an LP in a simulation using the null-message protocol, an LP in a TBS-based simulation does not need to know which other LPs can send it messages. Moreover, an LP in a TBS-based simulation can determine whether an event is executable *without waiting for information from other LPs.* The ability of an LP to make this determination on its own is what allows TBS-based simulations to scale well.

Example. Say that a logical process has the events E_{10}, E_{12}, E_{20}, and E_{22} in its event queue, where the subscripts indicate the events' timestamps, in simulated μs. We assume our logical process requires $4\mu s$ of real time to execute any event (this time corresponds to the time needed by whatever hardware is running the simulator). The logical process's clock, $Clock$, starts at zero. For simplicity, we assume that $s = 1$.

The event E_{10} becomes executable when $10\mu s < s \times t$. Because $s = 1$, the LP will execute this event after $10\mu s$ of real time have elapsed. Doing so takes $4\mu s$, after which, $Clock = 10\mu s$ and $t = 14\mu s$. The LP can then immediately execute E_{12}, since it can be sure that no messages with timestamps less than $14\mu s$ will arrive in the future. After executing this event, $Clock = 12\mu s$ and $t = 18\mu s$.

Now imagine that a message containing an event E_{21} arrives while the LP is executing E_{12}. The LP will place E_{21} into its event queue after it executes E_{12}, wait until $t = 20\mu s$, and then execute E_{20}, E_{21}, and E_{22}.

Consider what happens if executing E_{22} results in the transmission of an outgoing message, M, with timestamp T_{msg}. When the LP begins executing E_{22}, $t = 20\mu s + 2 \times 4\mu s = 28\mu s$ (since the LP must execute E_{20} and E_{21} before executing E_{22}). Therefore, M will arrive at its destination "on-time" (i.e., before it is executable) only if

$$T_{msg} \geq 28\mu s + t_{comp} + t_{latency}, \tag{3}$$

where t_{comp} is the time spent on computation before the LP can send M (this computation is a fraction of the total computation involved in executing E_{22}), and $t_{latency}$ is the latency to send M to the destination LP. For a general time scale, this equation becomes

$$T_{msg} \geq s \times (28\mu s + t_{comp} + t_{latency}). \tag{4}$$

A simulation designer can ensure that this inequality is true by decreasing the time scale.

From this example we can see two factors that can dramatically affect the performance of a TBS-based simulator. The first is the value of $t_{latency}$. Decreasing the latency to send a message between LPs enables a TBS-based simulator to use a higher time scale. The second factor is the ability of the LP to send its messages *as early as possible.* If, in our example, a programmer rewrote the simulator such that M was produced during the execution of E_{21} instead of E_{22}, the $28\mu s$ in Equation 4 would change to $24\mu s$, allowing us to increase our time scale.[1] Another point to note is that an LP does not have to execute an event as soon as the event becomes executable. For instance, in our example the LP executes E_{22} when the elapsed time is $28\mu s$, or $6\mu s$ after E_{22} becomes executable. Moreover, we should note the difference between the *simulation time*, or $Clock$, of a particular LP, and the elapsed *real time* for the entire simulation. Remember that, for a logical process LP_i, $Clock_i$ is equal to the timestamp of LP_i's most-recently-executed event. At any given real time, all of the LPs in a simulation can have different $Clock$s. On the other hand, the elapsed real time is a property of the entire simulation and is always equal for every LP. (It should be obvious that there will never exist a $Clock$ that is greater than $s \times t$.)

3. Building a TBS-based Simulator

In this section we describe how a typical PDES models a MANET with events, and we show how to modify

1 The ability to send messages early is similar to *lookahead* [9] in PDES.

such a typical simulator to take advantage of the properties of TBS and the NoC. Finally, we discuss how to estimate an upper bound on the time scale for a TBS-based simulation of a MANET, based on the characteristics of the simulated network and the NoC.

Modeling MANETs. Before describing our simulator, we give an overview of the way in which designers of discrete-event simulators use events to model wireless networks. We will also discuss the amount of simulated time between events. Given the real-time constraint of Equation 2, understanding how events are distributed throughout a simulated time line is important for someone designing a TBS-based simulator.

A MANET simulator typically uses two events per node to model a wireless transmission: The first represents the beginning of the transmission, and the second represents the end of the transmission. Consider an example with three nodes, node A, node B, and node C, which are simulated by LP_A, LP_B, and LP_C. Say that node A transmits a packet at simulated time $5\mu s$. The packet will reach nodes B and C only after some time, called the *propagation delay*, has elapsed. Assume that the propagation delay from node A to node B is $2\mu s$, and from node A to node C is $3\mu s$. The simulator will use three events to simulate the beginning of the transmission of the packet: one for node A with timestamp $5\mu s$, one for node B with timestamp $5 + 2 = 7\mu s$, and one for node C timestamp $5 + 3 = 8\mu s$. In a parallel simulator, LP_A will send the events for nodes B and C to LP_B and LP_C, respectively.

The second event at each node represents the end of the transmission. Say that the transmission of the packet lasts for $100\mu s$; LP_A, LP_B, and LP_C will schedule transmission-ending events with timestamps $105\mu s$, $107\mu s$, and $108\mu s$, respectively. Each logical process schedules its transmission-ending event *itself:* Neither LP_B nor LP_C would receive a message from LP_A telling it to simulate the end of the transmission. Instead, the initial messages contain fields indicating the duration of the simulated transmission.

In most MANET simulators, an LP performs the radio calculations for a given transmission, such as determining path loss and fading, when it executes the event representing the beginning of a transmission. If an LP executes an event representing the beginning of another transmission *before* simulating the end of the first transmission, then it must decide whether it should simulate a collision. This decision usually depends on the signal strengths of the two transmissions and some characteristics of the simulated receiver's radio.

Example. Suppose a node in our simulated MANET receives a data packet that contains some routing protocol information that it must use to update a table. Now suppose that the medium access control (MAC) protocol that the node is using dictates that the node must examine the data packet, wait for a $10\mu s$, and transmit an acknowledgment packet (ACK). Table 1 shows such a sequence of events taken from an actual MANET simulation. (Such sequences of events occurred frequently in the simulations described in Section 5.) When the LP executes SendAck it sends messages to the other LPs that simulate nodes receiving the ACK. (SendAck is analogous to E_{22} in the example in Section 2, since the execution of each event leads to its LP sending messages.)

Event	Time Since Previous Event
RadioBeginRx	n/a
RadioEndRxNoErrors	496.0
ExaminePacket	0
UpdateRoutingTables	0
CreateAck	0
SendAck	0
TransmitAckBegin	10.0
TransmitAckEnd	248.0

Table 1. The amount of simulated time between events in a simulation of a MANET. Times are in simulated μs.

The important point to note from this table is that the events are *not* distributed evenly. Most of the computation (other than radio calculations) in MANET simulations occurs between events that correspond to the end of transmissions and the events that correspond to the beginning of new transmissions. However, the simulated time between two such events is *much* shorter than the simulated time between the beginning and end of a transmission. In Table 1, for example, the transmission of the incoming data packet and outgoing ACK take 496 and 248 simulated μs, respectively, but there are only 10 simulated μs during which most of the computation occurs.

This distribution of events may at first make TBS seem like a poor event-synchronization protocol to use for simulating MANETs: If we slow down (decrease) the time scale such that 10 simulated μs scales to enough real time to do all of the necessary computation, then the periods of 496 and 248 simulated μs during which our LP is doing little computation will also scale, resulting in very long idle periods. Fortunately, by making a series of simple changes to the way in which a typical simulator executes events, we can create a relatively efficient TBS-based simulator. We will now discuss the process by which we arrived at this more-efficient simulator.

Initial Implementation of our Simulator. We began the development of our simulator by taking the code for a standard MANET simulator and simulating how well it would perform on the NoC using TBS (see Section 5 for a description of our simulation strategy). We shall describe the NoC hardware in Section 4; for now, it is sufficient to know that the NoC is a machine with enough processors such that we can have a one-to-one mapping between LPs and processors, and that these processors communicate by passing messages through a high-speed interconnect.

After performing our initial simulations, we quickly noticed that several critical paths limited the simulations' time scales. As we stated earlier, our simulator will execute correctly if every message arrives at its destination before its enclosed event is executable. The critical paths are naturally then the times between executing events that lead to one or more messages being sent, and latest time by which these messages can arrive at their destination processors without violating Equation 2. We shall now show how we were able to move computation off of these paths by changing the order in which our simulator would execute the events in Table 1. For the sequence of events listed in Table 1, the critical path is the time between executing RadioEndRxNoErrors and the arrival at their destination LPs of of the messages sent during the execution of SendAck. The left half of Figure 1 shows the unoptimized time line for executing this sequence of events.

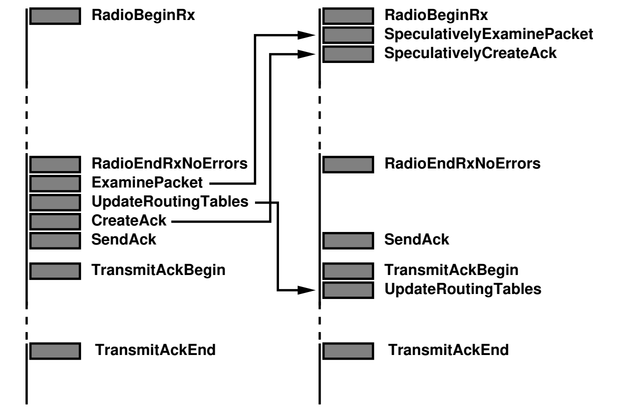

Figure 1. Moving computation off of our simulator's critical path.

Speculative Execution. Our first step in optimizing the way in which the simulator executes this sequence of events is to have the execution of ExaminePacket and CreateAck occur speculatively, after the execution of RadioBeginRx but before that of RadioEndRxNoErrors. The conventional simulator executes these events *after* RadioEndRxNoErrors to make sure that it will not receive any messages containing events corresponding to packets colliding with the original data packet. If such a simulated collision occurred, then the simulated data packet would have errors, the LP would execute RadioEndRxWithErrors, and it would merely simulate the receiving node dropping the packet. There would therefore be no ACK to simulate.

In our simulator, however, the processor will be idle for a long period of time between executing RadioBeginRx and RadioEndRxNoErrors. Therefore, if we speculatively execute ExaminePacket and CreateAck during this idle time and the LP eventually simulates a collision and the dropping of the packet, this speculative execution will not have cost us any time (it will have cost some energy, however). On the other hand, if the LP does not simulate a collision, then the processor will have less events to execute before SendAck than it would have had before our optimization. This makes our critical path shorter.

Postponed Execution. Likewise, the processor will be idle for a fairly long time after executing TransmitAckBegin. This period corresponds to the time spent simulating the transmission of the ACK. We can easily postpone the execution of UpdateRoutingTables to the time after TransmitAckBegin, since the content of the ACK does not depend on the updates to these tables.

After our optimizations, our final sequence of events looks like the following: RadioBeginRx, SpeculativelyExaminePacket, SpeculativelyCreateAck, RadioEndRxNoErrors, SendAck, TransmitAckBegin, UpdateRoutingTables, TransmitAckEnd (Figure 1). The path is now at the point where the processor will be able to send the messages corresponding to the transmission of the ACK almost *immediately* after RadioEndRxNoErrors becomes executable.

This example demonstrates the two guidelines we followed to optimize all paths like those in Table 1:

1. Perform speculatively whatever computation may influence *the next outgoing message.*
2. Postpone whatever computation is not necessary to form a given message to the time *after* sending the message, when the LP will be simulating the transmission of the packet that the message represents.

Determining the Time Scale. To choose a value for s, we rewrite Equation 2. If we assume that the event leading to the sending of messages (in our example, RadioEndRx-

NoErrors) is able to be executed as soon as it becomes executable,[2] our constraint for correctness becomes:

$$\Delta T \geq s \times \Delta t, \tag{5}$$

where ΔT is the difference between the timestamp of the current event and the timestamp of the final message sent as a result of executing this event, and Δt is the real time between the event in question becoming executable and the first word of the last message reaching its destination. ΔT depends only upon the simulated MANET: It is the sum of the time, called the **transmitter-turn-on time (TTOT),** for a node's radio to change from sensing mode to transmitting mode, and the worst-case (longest) **propagation delay** between the sending node and one of the receiving nodes. We will call these two times T_{ttot} and T_{prop}.

Δt depends on two factors: the time the NoC processor needs to execute the instructions that will send all of the messages into the interconnect, and the worst-case latency for the last message sent into the interconnect to reach its destination processor. The latter is a function of the NoC itself; we will refer to it as t_{lat}. The former is essentially the product of the number of messages to be sent, the number of bytes per message, and the time required by the processor to send one byte into the interconnect.

We can perform one final optimization that eliminates the dependence of Δt upon the length of messages. For each message a NoC processor would normally send, we introduce an additional *reservation message,* which contains only the timestamp of the original message (we will now refer to the original message as the *full message*). When a NoC processor simulates the transmission of a packet, it first sends reservation messages to all of the receiving processors, and then sends the full messages.

When an LP receives a reservation message, it knows that it will soon receive a full message, and so it does not execute any events with timestamps later than the reservation message's timestamp. This scheme ensures that all LPs will still execute events in order, while sending only reservation messages, which have minimal length, during the critical path.

We say that Δt is equal to $t_{lat} + t_{send} \times n$, where t_{send} is the time for the processor to send one reservation message into the interconnect and n is the number of messages per simulated transmission. Using this expression, plus our equation for ΔT, Equation 5 becomes

$$T_{ttot} + T_{prop} \geq s \times (t_{lat} + t_{send} \times n), \tag{6}$$

which we can rewrite:

$$s \leq \frac{T_{ttot} + T_{prop}}{t_{lat} + t_{send} \times n}. \tag{7}$$

2 Simulations have shown that this assumption is almost always true.

The IEEE 802.11 MAC protocol [10] specifies T_{ttot} as $5\mu s$. If we take a worst-case value for T_{prop} of zero (since mobile nodes may be very close to one another) and for n of 32, and we let t_{send} be 8 ns and t_{lat} be 100 ns [14] (we expect actual values from the NoC to be similar to these), then the right-hand side of Equation 7 becomes approximately 14.0, meaning that our simulator should be able to simulate MANETs with these parameters *fourteen times faster than real time.* Moreover, the execution time is *independent of the size of the simulated MANET,* as long as n remains constant.

4. The Network on a Chip

In this section we describe how we have designed the NoC to efficiently execute a TBS-based MANET simulator. The presence of the NoC is critical to the performance of our simulator; other parallel computing platforms would not be able to execute our simulator at a reasonable time scale [14].

In our simulator, there is a one-to-one mapping between simulated network nodes and LPs, and a one-to-one mapping between LPs and processors in the hardware executing the simulator. Therefore, a machine executing our simulator must have thousands of processors. Moreover, because the performance of our simulator depends heavily on the latency to pass messages between LPs, we need a machine that allows processors to communicate efficiently. Currently-existing parallel platforms that a person would consider for running a TBS-based MANET simulator therefore include distributed shared-memory (DSM) machine and networks of workstations (NoWs). Unfortunately, the largest DSM machines contain only 1024 processors [16], making them incapable of running TBS-based simulations of MANETs containing greater than 1024 nodes. NoWs are more promising: NoWs containing more than 1024 nodes certainly exist [8], and the message-passing latency in a NoW can be as low as $6.3\mu s$ [23]. Unfortunately, however, a NoW is still a bad fit. TBS-based simulation of MANETs is an application with a large ratio of latency-critical communication to computation. Running such an application on a NoW containing thousands of very-powerful computers is a poor use of resources: The simulation's time scale, and therefore the performance of the simulator, will be limited by the latency to pass messages between workstations. (If we perform the same calculation as we did at the end of Section 3, but with t_{lat} equal to $6.3\mu s$, we get an upper bound of 0.76 on s.) We would prefer instead a platform with less-powerful computers but a lower message-passing latency.

In addition to having thousands of moderately-powered processors that can communicate quickly, a machine that executes our simulator should have processors that can efficiently manage event queues. To do this, the processors

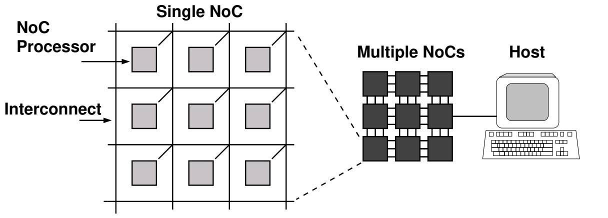

Figure 2. A multi-chip NoC simulator and host.

need a low-overhead mechanism for determining when an event has become executable: They must be able to quickly compare the timestamps of scheduled events with the scaled version of elapsed time.

With these requirements in mind, we created the NoC [14, 15], a chip multiprocessor. We estimate each chip will contain approximately 100 processors. To enable simulations of MANETs containing thousands of nodes, we have designed the NoC such that we can gluelessly combine multiple chips to create a massively-parallel machine. The NoC processors are designed *specifically* to execute LPs in our simulator, thus they lack virtual memory or any other hardware operating-system support. Each processor has its own private 8KB memory, and communicates with the other processors only by passing messages via a highly-pipelined interconnect.

A simulation run on the NoC is managed by on an off-chip workstation called the *host.* The host can send and receive messages to and from the NoC processors; it sends the processors the code they will execute during a simulation and it collects statistics when a simulation is complete. Figure 2 shows a multi-chip NoC simulation connected to a host.

The NoC processors lack multiply / divide and floating point units, meaning that they would require a great deal of time to perform complicated radio calculations. Therefore, instead of the NoC processors performing radio calculations *during* a simulation, the host performs the calculations *before* the simulation begins; it can do so for any simulation in which the movement patterns of the nodes are known before the simulation begins (for example, static networks, or networks using the random-way point [12] mobility model). The host incorporates these calculations into the code that it sends to the processors (the code for the radio layers has a section that is different for each processor; it tells a given processor how to simulate incoming transmissions based on the sources of the transmissions and the times at which they occur).

Because the processors execute LPs, which in turn simulate network nodes, a processor that simulates a given network node will need to exchange messages only with processors simulating network nodes within its node's transmission range. Therefore, if the simulation user maps LPs to processors in an intelligent way (i.e. processors that are close together on a chip simulate nodes that are close together in the simulated terrain), no messages should travel more than a few hops through the interconnect. In simulations of networks with highly-mobile nodes, a mapping that is efficient at the beginning of a simulation may become quite inefficient later. Such simulations can be paused, re-mapped, and started again by the host. In simulations in which the mobility patterns of the nodes are known before the simulation begins, the host can precompute the remappings. A researcher using the NoC to simulate a network with high node mobility may wish to make remapping easier by using only a fraction of the NoC processors on each chip.

Every processor has a *timer coprocessor,* which it uses to schedule new events, and which alerts it when a previously-scheduled event becomes executable [15]. To determine when an event is executable, the timer coprocessors need to keep track of the current elapsed time; for this purpose, they each contain an *incrementer.* All incrementers start from zero on reset and change their values at the same rate. Thus, every processor has the same notion of the current elapsed time.[3]

Because all of the incrementers advance independently (though at the same rate), there is no centralized control governing all of the processors in the NoC. Hence, adding more nodes to our simulated MANET, and therefore more processors to our simulator, does not add any extra hardware synchronization costs. This ability of the hardware to scale well makes possible the fast simulation of large-scale MANETs.

5. Evaluation

In Section 3 we showed that the performance of a TBS-based simulator running on the NoC is independent of the size of the simulated MANET, as long as parameters such as the propagation delay and the density of the network do not vary with the number of nodes. We have therefore taken a problem—simulating a given class of MANETs—and found a solution with $O(1)$ running time. [4] (By "class of MANETs" we mean a collection of MANETs of different sizes but with common values for many other characteristics such as TTOT, density, MAC and routing layer protocol, and propagation delay. In other words, we expect that, for a given class of MANETs, the average amount of computation to simulate a node remains constant as N changes.)

3 If all of the incrementers do not have precisely the same value, the simulator can still execute correctly [14]. In this case, we consider the t from Equation 2 to be the *receiving processor's* notion of the current elapsed time. Hence, we can compensate for the difference between incrementers by decreasing the time scale by an appropriate amount.

4 Our solution does use $O(N)$ processors, however.

A sequential simulator is a solution with, at best, $O(N)$ running time, where N is the number of nodes in the simulated network.

An $O(1)$ solution to a problem will naturally outperform an $O(N)$ solution once N becomes large enough. We need to determine, however, what value of N is "large enough" for simulating various classes of MANETs. If, for example, our $O(1)$ solution has such a large constant factor attached to its run time that it outperforms a sequential simulator only when N is on the order of ten million, then it will be far less useful than if it surpasses the sequential simulator when N is a hundred or a thousand. In this section we will show that our TBS-based simulator is faster than a sequential simulator for MANETs containing a hundred nodes. We will also discuss how some characteristics of the simulated MANETs affect the magnitude of the TBS-based simulator's speed advantage.

Simulating Our Simulator. To determine the time needed for a TBS-based simulation running on the NoC, we need to know only the time scale of the simulation. For the results in this section, we used Equation 7 to estimate the time scale for a given simulation, based on the worst-case value of n (the number of messages sent per simulated transmission). To verify that this time scale was safe (i.e. that we did not violate our condition for correctness, Equation 2), we developed a program called `nocsim`.

We modified a sequential MANET simulator such that it creates log files listing, for every simulated node, every event (and corresponding timestamp) executed during a given simulation. The log file for a particular node corresponds to the code that would be executed by the NoC processor simulating the node in our TBS-based MANET simulator. `nocsim` uses these logs, along with estimates of the time to execute events and to send messages through the NoC interconnect, to check whether such a simulation would execute correctly using the time scale we derived from Equation 7.

We obtained our estimates of the time to execute events by compiling for the MIPS ISA [13] (which is similar to the NoC processors' ISA [14]) the event-execution code segments from the sequential simulator, and running them in a MIPS simulator to obtain worst-case instructions counts (because the NoC processors do not have caches, memory access time is uniform). We turned these instruction counts into times by assuming 500 MHz processors (this is conservative for the process, TSMC $0.18\mu m$, that we shall use for the NoC). We assumed a worst-case delay of 100 ns to send a message; this delay is great enough to include the time for a message to cross one inter-chip boundary [28]. We assumed the time to send one reservation message into the NoC interconnect was 8 ns.

Results. We now present simulation results that show how various characteristics of a simulated MANET influence the performance of the TBS-based simulator, relative to that of the sequential simulator that we used to produce the log files for `nocsim`. The scenarios we simulated are similar to those used in [2] and [5]. We used networks consisting of 100 nodes using the IEEE 802.11 Distributed Coordination Function (DCF) MAC layer [10] and the Ad-hoc On-Demand Distance Vector (AODV) routing protocol [24]. The channels have a bandwidth of 2 Mbps. A fraction of the total nodes were constant-bit-rate sources, a fraction were sinks, and the rest forwarded packets between the sources and sinks. The nodes used a standard random-waypoint mobility model [12]. We used a two-ray path loss model and a simple radio model that does not take into account acoustic noise. We varied the density of the networks by adjusting the size of the simulated terrain.

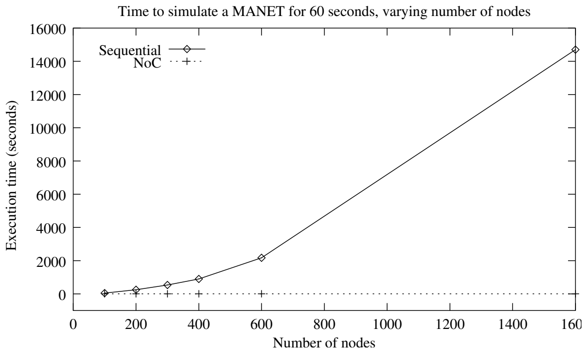

Figure 3. Execution times of the sequential simulator and the TBS-based simulator running on the NoC, various numbers of nodes.

Figure 3 shows the actual execution time of our sequential simulator and the projected execution time of the TBS-based simulator running on the NoC for MANETs of various sizes. The different simulated networks have all of the same characteristics (density, fraction of nodes serving as sources or sinks, packet size, etc.), and only vary in size. From this figure we can see that the TBS-based simulator is faster than the sequential simulator for networks containing as few as a hundred nodes, and that the disparity between the speeds of the two simulators increases dramatically as we increase the number of nodes in the simulated network.

Figure 4 shows the execution times of the sequential and TBS-based simulators for networks containing 100 nodes for various densities. The x axis shows, for a particular MANET, the largest number of messages that any NoC processor had to send to simulate a single transmission. This number corresponds to the density of the MANET—in a

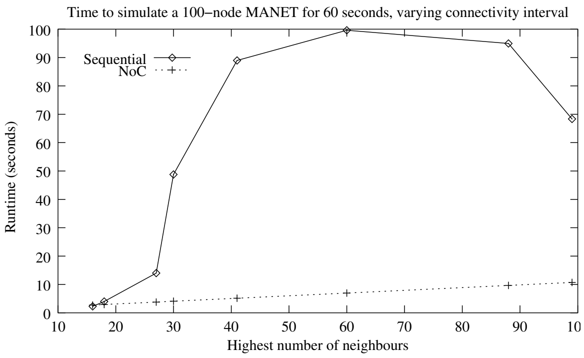

Figure 4. **Execution times of the sequential simulator and the TBS-based simulator running on the NoC, various densities.**

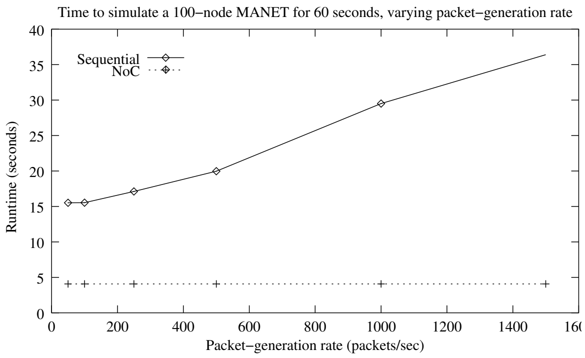

Figure 5. **Execution times of the sequential simulator and the TBS-based simulator running on the NoC, various packet-generation rates.**

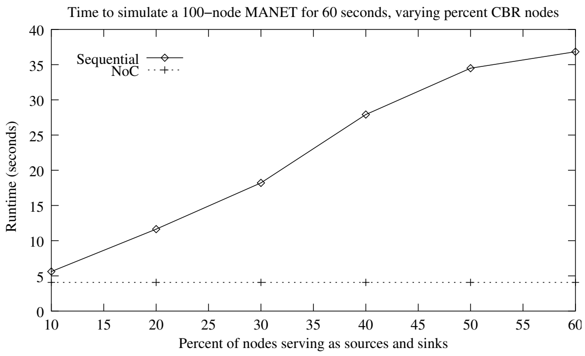

Figure 6. **Execution times of the sequential simulator and the TBS-based simulator running on the NoC, various percentages of nodes serving as CBR sources and sinks.**

simulation of a denser MANET, each NoC processor will typically have to send more messages to simulate a transmission. In these scenarios, 50% of the simulated nodes are constant-bit-rate (CBR) sources or sinks (simulations of networks with different fractions of nodes acting as sources and sinks yielded similar results). The sources generate two 512-byte packets every second.

There are two points to note from Figure 4. First, the projected execution time of the TBS-based simulator is superior to that of the sequential simulator, even for networks containing only 100 nodes. As we increase the number of nodes (while keeping the density the same), this disparity will increase, since the execution time of the sequential simulator will scale, at best, linearly [30], while the execution time of our simulator will remain constant, even for networks containing tens of thousands of nodes.

The second point to note is that the TBS-based simulator's advantage over the sequential simulator decreases when simulating MANETs that are extremely dense or sparse. In extremely sparse networks, many nodes are inactive; the sequential simulator therefore has less computation to perform, so it performs relatively well. The high connectivity of extremely dense networks means that simulating some transmissions requires a large number of messages (sometimes as many as 99), thus limiting the TBS-based simulator's time scale (see Equation 7). The MANETs for which our simulator performed the best are those in which all processors are busy, yet connectivity remains fairly low.

Figure 5 shows the execution times of the two simulators as we vary the rate at which the simulated CBR sources generate packets. Changing the packet-generation rate is similar to changing the density in that higher rates, like higher densities, lead to busier processors. However, this increase in busyness comes without an increase in connectivity. Hence, the advantage of the TBS-based simulator increases monotonically with higher packet-generation frequency.

Likewise, Figure 6 shows how changing the percentage of nodes serving as CBR sources and sinks (all of the sources are producing one 512-byte packet per second) affects the relative performance of the two simulators. The results in this figure are similar to those in Figure 5, since increasing the number of nodes serving as sources or sinks results in a nondecreasing amount of work for each processor in the TBS-based simulator without changing the connectivity of the simulated MANET.

The projected execution times for the TBS-based simulator shown in Figure 5 and Figure 6 are constant: Changing the packet-generation rate or the percentage of nodes acting as sources or sinks does not change any of the parameters used by Equation 7 to determine the time scale. The time scales for Figure 4, on the other hand, varied with the worst-case number of messages per transmission. The highest time scale that was twenty two, and the lowest was four. Note, however, that the TBS-based simulator's best performance, relative to the sequential simulator, did *not* come when it used the lowest time scale; the MANETs which dictated such low time scales also very little activity, allowing the sequential simulator to simulate them very quickly.

The time scales used for the MANETs presented in this section should work for larger MANETs, as long as the MANETs' density remains the same. We verified this for many classes of MANETs by creating log files for MANETs containing thousands of nodes and using `nocsim` to check that Equation 2 was true for every message.

6. Related Work

Most researchers studying MANETs simulate networks with ns-2 [21], Glomosim, Opnet [7], or QualNet. Of these, only Glomosim and QualNet are parallel simulators, both of which use conservative event-synchronization protocols. SWAN is another recently-developed conservative parallel simulator.

The authors of [31] show the speedup over sequential execution of parallel Glomosim simulations of MANETs with at most 3,000 nodes. The greatest speedup in the paper is a factor of nine, with sixteen processors, using an IBM 9076 SP (a distributed-memory multicomputer). Similarly, [1] contains the results of simulations of wireless networks up to 3,000 nodes, with a maximum speedup of slightly less than eight, on sixteen processors.

QualNet is the commercial successor to Glomosim. The creators of QualNet report speedup of 12 with a 16-processor machine running a 10,000-node model [26].

The authors of [20] use SWAN to simulate a network with 10,000 nodes using a simplified MAC-layer model. They report that simulating 1,000 simulated seconds took more than ten hours to complete with five processors. The paper does not include speedup results. The results of [19] show speedup of five when using eight processors.

As we have shown in Section 5, our simulator (when run on the NoC) should be able to simulate networks like those simulated in the papers cited above many times faster than real time. This ability results from the excellent scaling abilities of the hardware and the software.

The most commonly-cited event-synchronization protocols are the optimistic Time Warp [11] protocol and the conservative null-message protocol. Further descriptions of synchronization protocols are found in [9] and [29].

Our approach is somewhat similar to the technique of network emulation. Examples of network emulation include `dummynet` [25].

7. Summary

In this paper we have presented a new synchronization protocol, TBS. We have shown how to design a TBS-based MANET simulator for execution on a highly-parallel message-passing machine, and we have demonstrated that our combination of hardware and software is capable of simulating large-scale networks faster than real time.

In the future, we plan to evaluate how well simulators using conventional event-synchronization protocols, such as the null-message protocol or Time Warp, would execute on a parallel machine like the NoC (such simulators would have no need for the timer coprocessor). We also plan to design and evaluate a TBS-based MANET simulation in which a logical process can simulate more than one node, or in which more than one logical process can run on a single computer. If, for example, we used either of these new mapping techniques to simulate a hundred nodes on one processor, then hundred-workstation NoW could simulate a 10,000-node MANET. The longer latency to pass messages in a NoW and the affects of simulating more than one node per processor would naturally make such a simulation slower than a simulation using a 10,000-processor collection of NoC chips, but such a simulator would not require any custom hardware, and might still be faster than a simulator using a conventional event-synchronization protocol.

We also plan to explore the possibility of a new class of MANET routing protocols that take advantage of the faster-than-real-time speed of our simulator. Imagine, for instance, a MANET in which some of the nodes contain NoCs. During the life of the MANET, if such a node faced a choice—such as whether to move north or south, for example—it could use its NoC to quickly predict the effects of both options on the future behavior of the network and then make the best choice.

Acknowledgments

This work was supported by the Multidisciplinary University Research Initiative (MURI) under the Office of Naval Research Contract N00014-00-1-0564.

References

[1] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H. Song. PARSEC: A Parallel Simulation Environment for Complex Systems. *IEEE Computer*, **31**(10):77–85, October 1998.

[2] J. Broch, D. Maltz, D. Johnson, Y. Hu, J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Nework Routing Protocols. In *Proc. of the ACM/IEEE MobiCom,* October 1998.

[3] R. Bryant. Simulation of Packet Communications Architecture Computer Systems. MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[4] K. Chandy and J. Misra. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Communications of the ACM 24,* 11 (November 1981), 198-205.

[5] S. Das, C. Perkins, E. Royer, and M. Marina. Performance Comparison of Two On-demand Routing Protocols for Ad hoc Networks. *IEEE Personal Communications Magazine special issue on Ad hoc Networking,* February 2001, p. 16-28.

[6] DaSSF: The Darthmouth Scalable Simulation Framework. http://www.cs.dartmouth.edu/research/DaSSF/.

[7] F. Desbrandes, S. Bertolotti, and L. Dunand. Opnet 2.4: an environment for communication network modeling and simulation. *Proceedings of the European Simulation Symposium*, pp. 609–614, Delft, Netherlands, October 1993.

[8] The Earth Simulator: System Configuration. http://www.es.jamstec.go.jp/esc/eng/Hardware/system.html.

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

[10] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[11] D. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, **7**(3):404–425, July 1985.

[12] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing,* edited by Tomasz Imielinski and Hank Korth, Chapter 5, pages 153-181. Kluwer Academic Publishers, 1996.

[13] G. Kane and J. Heinrich. Mips Risc Architecture. Prentice Hall. 1991.

[14] C. Kelly. Wireless Network Simulation Done Faster Than Real Time. Master's Thesis, Cornell University, Ithaca, NY, 2002. Available from http://www.csl.cornell.edu/∼clint/.

[15] C. Kelly, V. Ekanayake, and R. Manohar. SNAP: A Sensor Network Asynchronous Processor. *Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems,* Vancouver, BC, May 2003.

[16] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. *ISCA 1997*: 241-251.

[17] B. Leiner, R. Ruth, and A. Sastry. Goals and Challenges of the DARPA GloMo Program. *IEEE Personal Communications,* 3(6):34-43, December 1996.

[18] M. Liljenstam, R. Ronngren, and R. Ayani. MobSim++: Parallel Simulation of Personal Communication Networks. *IEEE DS Online 2,* 2.

[19] J. Liu and D. Nicol. Lookahead revisited in wireless network simulations. *Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS'02)*, pages 79-88, May 2002.

[20] J. Liu, L. F. Perrone, D. M. Nicol, M. Liljenstam, C. Elliott, and D. Pearson. Simulation modeling of large-scale adhoc sensor networks. *European Simulation Interoperability Workshop,* 2001.

[21] S. McCanne and S. Floyd. The ns network simulator. http://www.isi.edu/nsnam/ns/.

[22] R. Meyer and R. Bagrodia. Path Lookahead: A Data Flow View of PDES Models. *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS '99),* May 1-4, 1999 in Atlanta, Georgia.

[23] Myrinet Performance. http://www.myri.com/myrinet/performance/

[24] C. Perkins and E. Royer. Ad-hoc On-Demand Distance Vector Routing. *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications,* New Orleans, LA, February 1999, pp. 90-100.

[25] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review,* Jan. 1997.

[26] Scalable Network Technologies. Qualnet. http://www.scalable-networks.com/.

[27] Sivakumar, Sinha, Bharghavan. CEDAR: A Core-Extraction Distributed Routing Algorithm. *IEEE Journal on Selected Areas in Communications.* Vol 17, No. 8, August 1999.

[28] J. Teifel. Interchip Communication in Asynchronous VLSI Systems. *Cornell Computer Systems Lab Technical Report CSL-TR-2002-1027*, October 2002.

[29] V. Vee and W. Hsu. Parallel Discrete Event Simluation: A Survey. Technical Report, Centre for Advanced Information Systems, Nanyang Technological University, Singapore, August 1999.

[30] K. Walsh, E. Sirer. Staged Simulation for Improving the Scale and Performance of Wireless Network Simulations. *Winter Simulation Conference,* New Orleans, LA, December 2003.

[31] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks. *Proceedings of the 12th Workshop on Parallel and Distributed Simulations—PADS '98*, Banff, Alberta, Canada, May 1998.