Efficient Parallel Simulation of Mobile Wireless Networks by Run-time Prediction of Multi-hop Propagation Delay

Hirozumi Yamaguchi, Kazuki Konishi and Teruo Higashino
Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871 Japan
{h-yamagu, higashino}@ist.osaka-u.ac.jp

***Abstract*— In this paper, we propose a method to improve the performance of parallel and distributed simulation of multi-hop mobile wireless networks. We assume that the simulation field is divided into several pieces and is simulated by multiple parallel processes which synchronize conservatively, *i.e.*, without violation of causality. The basic idea of our approach is to predict the lower bounds of timestamps of "potential" upcoming events which cause inter-process messages based on the timestamps of the currently scheduled events plus minimum multi-hop propagation delay estimated in run-time. By this prediction method, we are able to enlarge the lookahead of parallel processes, and their concurrency can be enhanced. We have implemented the parallel and distributed version of our network simulator MobiREAL, and have also implemented this mechanism. The experimental results have shown that we could achieve 50% speed up in average compared with the version using static lookahead in the simulation of end-to-end communications over 1,200 nodes.**

I. Introduction

Discrete Event Simulation (DES) is a simulation technique that schedules and executes timestamped events. A discrete event simulator has an event queue in which events are sorted in the incremental order of timestamps. The simulator dequeues the event with the earliest timestamp, proceeds the simulation clock to the timestamp, and executes the event. Then subsequent new events are scheduled with appropriate timestamps. This traditional technique has been widely used, and most network simulators are also discrete event simulators. Parallel Discrete Event Simulation (PDES) [1], [2] is performed by cooperative multiple DES simulators (called *Logical Processes (LPs)* hereafter). PDES is required to pursue maximum parallelism for speed up, satisfying causality of events. To achieve this goal, several synchronization protocols have been considered. These protocols are classified into two categories, conservative and optimistic synchronizations.

Conservative synchronization protocols never violate causality at any time. Thus when an LP processes an event, it must be ensured that there is no event which has an earlier timestamp than that of the event being processed and is not yet processed. A well-known protocol of this category is the NULL message algorithm by Chandy and Misra [3]. On the other hand, in optimistic synchronization protocols each LP proceeds simulation without knowing the other LPs' events. If an LP receives a message that notifies an event with past timestamp, LPs recognize the violation of causality, and a rollback is performed. This method is helpful to maximize concurrency but the protocol itself is rather complicated and the simulator incurs overhead by the rollback mechanism. Time Warp algorithm [4] is one of the protocols of this category.

In either of the two synchronization schemes, messages are exchanged among LPs that notify events and their timestamps to each other. Let us assume that an event in an LP yields subsequent events in other LPs. For example, if a wireless signal transmission is performed by a node allocated to LP i, then subsequent signal receptions are performed by its neighborings nodes. If those neighboring nodes are not allocated to LP i, then the reception events and their timestamps are notified by LP i using "inter-process" messages (called *event messages* hereafter). Those LPs which receive the event messages are called *neighbors* of LP i.

In the conservative synchronization scheme, *lookahead* largely affects the performance of simulation [1]. Intuitively, lookahead is the time length during which each LP can proceed simulation independently of the other LPs. More formally, lookahead is defined as follows. For LP i, the earliest timestamp in the event messages sent from the neighboring LPs to LP i is called the Earliest Input Time and is denoted by EIT_i. Similarly, the earliest timestamp of events in the event messages sent from LP i is called the Earliest Output Time and is denoted by EOT_i. From these definitions, these two values must satisfy the following equation at

any time.

$$EIT_i = min_{j \in \{neighbors\ of\ i\}} EOT_j \quad (1)$$

To keep causality, each LP i is allowed to process its own events which have the earlier timestamps than EIT_i. We let T_j denote the current simulation time of LP j. The lookahead of LP j, denoted by LA_j, is the time duration, where LP j is allowed to process events with earlier timestamps than $T_j + LA_j$. Therefore, at any time, EOT_j is determined by the lookahead, as described below.

$$EOT_j = T_j + LA_j \quad (2)$$

Thus lookahead is essential to the performance of PDES.

In this paper, we propose a method to improve the performance of parallel simulation of multi-hop mobile wireless networks based on PDES with conservative synchronization. We assume that the simulation field is divided into several pieces and is simulated by multiple parallel processes which synchronize conservatively, *i.e.*, without violation of timestamp causality. Considering the characteristics of multi-hop mobile wireless networks, we predict the lower bounds of timestamps of potential upcoming events by estimating minimum multi-hop propagation delay. This prediction is done in run-time by seeking all the currently scheduled events and geographic location of these events. Unlike the former methodologies which have determined lookahead by link latency or delay in protocol stacks which can be precomputed before simulation, we compute the lookahead online. Thus the value of lookahead is different at each moment. On the other hand, we may suffer from additional cost to compute lookahead. Therefore, we provide a method where lookahead can be calculated on-the-fly based on the location of scheduled events. We have implemented the parallel and distributed version of our network simulator MobiREAL [5]–[7] and have also implemented this mechanism. The experimental results have shown that we could achieve 50% speed up in average compared with the version using precomputed lookahead in the simulation of end-to-end communications over 1,200 nodes.

II. Related Work and Contribution

A. Related Work on Parallel Wireless Network Simulation

Regardless of wired or wireless networks, the most simplest lookahead is propagation delay between two nodes allocated to different LPs. For example, if a node x on LP i generates a "sending event" that sends a packet at time t to node y on LP j, a corresponding "receiving event" that notifies the arrival of this packet to node y is scheduled with timestamp $t + d$. However, d is usually small and without intelligent prediction we cannot expect lookahead which is large enough.

Several researchers have focused on reducing overhead in optimistic synchronization. WiPPET [2], [8] uses the Georgia Time Warp (GTW) simulator, the general purpose optimistic parallel simulator [9]. SWiMNet [10] is also a parallel wireless network simulator which mainly aims at reducing rollbacks in optimistic synchronization.

Meanwhile, many research efforts have been dedicated for better efficiency of PDES by improving lookahead [11]–[13] in conservative synchronization. In Ref. [11], lookahead is determined by the delay in the MAC and PHY layers. This mechanism has been implemented in GloMoSim and its commercial version, QualNet [14]. In particular, they focus on Inter Frame Space and backoff time in IEEE802.11 MAC. In Ref. [12], three types of lookahead calculation for wireless ad hoc network are presented; data transit delay in node entities, delay from the protocol behavior, and delay by channel contention. This mechanism has been implemented in SWAN simulator. In Ref. [13], a lookahead prediction method called path lookahead is presented where "communication path delay" through different layers are considered.

B. Our Contribution

Even though we have the same goal as the above methodologies like Refs. [11]–[13] which aim at improving lookahead in conservative synchronization, our methodology is original and new in the following points. First, we focus on *multi-hop propagation delay* in mobile wireless networks. We take the location and mobility of nodes into account to estimate the earliest timestamps of the events to be scheduled in the neighboring LPs. Secondly, to enable the estimation of such multi-hop propagation delay which differs in time and location, we present a technique to *compute lookahead in run-time*. When an LP needs to estimate EOT, the LP predicts the upcoming events, which are not yet scheduled in its event queue. Based on the prediction, the LP knows the future events that will yield subsequent events in the neighboring LPs and thus has occasion to determine larger lookahead transiently. As far as we know, most of the existing methodologies exploit the delay incurred in protocol stacks and/or radio propagation to improve lookahead, and hence we believe that the presented idea is original. Obviously the technique incurs additional computation cost of predicting upcoming events. Therefore, we provide a prediction algorithm which is lightweight but effective enough. In the experiments we

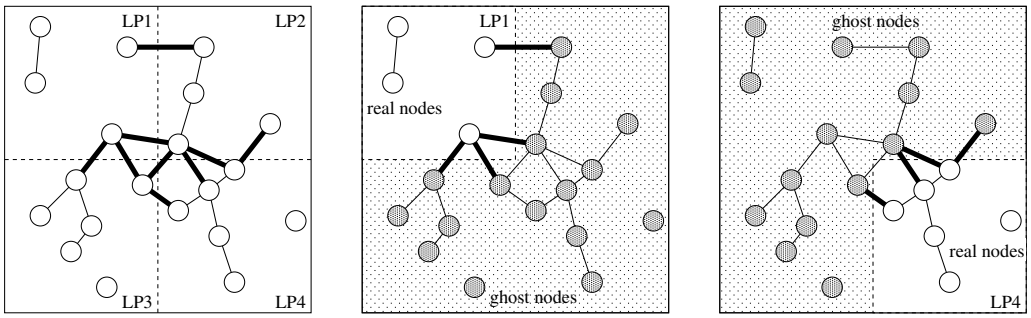


Fig. 1. Field Division

confirmed that the total performance gain by improved lookahead dominated the disadvantage of incurring computation costs.

Finally we would like to mention that this technique can co-exist with the existing techniques that exploit static lookahead. That is, the technique does not compete with those techniques, and thus can be implemented in many parallel discrete event wireless network simulators.

III. Improving Lookahead by Run-time Prediction of Multi-hop Propagation Delay

We assume that a simulation field is a form of a rectangle, and the field is divided into multiple pieces of rectangles. Each LP manages one of these pieces, and simulates the behavior of network nodes on that field. Hereafter, for an LP, the objects of nodes allocated to the LP are called *real nodes*. For the other nodes, the LP generates dummy objects called *ghost nodes*. Fig. 1(a) shows an example where the field is divided into four pieces. Each LP manages one-fourth of the field like LP1 in Fig. 1(b) and LP4 in Fig. 1(c). We note that the thin lines between nodes represent logical wireless connectivity and each bold line indicates that any communication over the line yields event messages between LPs. Our key idea to improve lookahead is to predict the occurrence of such cross-border communication as early as possible.

A. Outline of Run-time Prediction

Hereafter, an event that represents the start of frame transmission in the physical layer is called a *physical transmission event*. An event is said to be a *special event* iff it will yield a subsequent physical transmission event. An example of special event is an application packet transmission event. The sender in the physical transmission event caused by a special event se is called the *origin node* of se and is denoted by $se.org$.

If LP i finds a special event se when seeking the event queue at a simulation time T_i, it estimates the minimum number of hops from $se.org$ to a ghost node. We assume that for each number k of hops, the k-hop minimum propagation delay can be estimated in advance (two algorithms to estimate this delay will be given in Section III-C). By seeking all the future special events and predicting the future transmission events, LP i can determine larger EOT_i, the earliest timestamp of the events to be scheduled on another LP, say j. As a result, this may enlarge the lookahead of LP j.

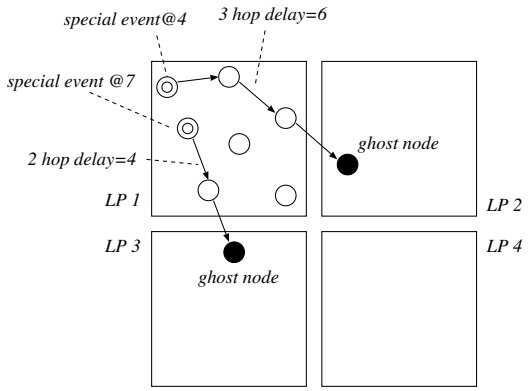
Formally, we let $se.hop$ and $se.ts$ denote the estimated minimum number of hops and the timestamp of special event se, respectively. Also the estimated k-hop minimum propagation delay is denoted by $T_H(k)$. Then EOT_i is given below.

$$EOT_i = min_{se \in SE_i[T_i, T_i']}\{se.ts + T_H(se.hop)\} \quad (3)$$

$SE_i[p, q]$ denotes the set of special events in LP i whose timestamps are in the range $[p, q]$. Here, clearly LP i does not need to seek the special events with timestamps later than EOT_i. Thus we may set the upper bound of EOT_i to T_i'. Also, for safe synchronization, the upper bound of EOT_i needs to be $EIT_i + T_H(1)$. The reason is as follows. According to the definition of EIT_i, at time EIT_i, a signal reception event may be scheduled at a border node, and in response a signal transmission event may be scheduled at the same node. In this case, clearly EOT_i is EIT_i plus one-hop minimum propagation delay. Therefore, $T_i' = EIT_i + T_H(1)$.

Fig 2(a) shows an example. We assume $T_i = 3$, $EIT_i = 10$ and $T_H(k) = 2k$. Then $EOT_i = \min\{10 + 2,\ 4 + 6,\ 7 + 4\} = 10$. We note that computation of $T_H(k)$ is given in more details in Section III-C. Fig 2(b) shows the algorithm description.

Finally we note that from equation (2), lookahead decreases as the time passes, but accordingly lookahead is updated as the events are processed.



(a) Example

$EOT_i = EIT_i + T_H(1)$
`while` // seek special events with timestamps in $[T_i, EOT_i]$
  $se = getEarliestSpecialEvent()$
  `if (` $se = null$ `or` $se.ts \geq EOT_i$ `) break_while endif`
  $EOT_i = \min\{ EOT_i,\ se.ts + T_H(se.hop) \}$
`endwhile`

(b) Algorithm Description

Fig. 2. Lookahead Computation

B. Estimating Minimum Number of Hops

To estimate the minimum number $se.hop$ of hops for special event se, several methods can be considered. Most accurately, we may build topology snapshots and find the minimum number of hops, but this may have serious impact on the performance. We propose two different methods for estimating the minimum number of hops. One focuses on reasonable computational cost, while another pursues more accuracy. The former and latter methods are called *Cost Aware Hop Computation* (CAHC method in short) and *Precise Hop Computation* (PHC method in short), respectively.

1) Cost Aware Hop Computation: The CAHC method uses the minimum Euclid distance from the location of $se.org$ to the border of the field. The minimum Euclid distance from node u to the border at time T_i is denoted by $D_{u@T_i}$. $D_{u@T_i}$ can be computed easily since the field is rectangle. We also let $N(k)_{@T_i}$ denote the set of all the nodes on an LP which need *at least* k-hops to reach ghost nodes at time T_i. Additionally, we introduce the following notations; the maximum radio range R_{max}, the maximum speed V_{max} of nodes, and the (estimated) maximum lookahead LA_{max} throughout the simulation. Then $N(k)_{@T_i}$ is calculated as follows in the CAHC method.

$$N(k)_{@T_i} = \begin{cases} \textit{all the real nodes on the LP} & (k=1) \\ \{u \mid D_{u@T_i} - V_{max} * LA_{max} \geq (k-1) * R_{max}\} & (k>1) \end{cases} \quad (4)$$

Each node in $N(k)_{@T_i}$ is still at least $(k-1) * R_{max}$ distance away from the border even after it moves toward the border in the maximum speed during the time of maximum lookahead. This means that at least k-hops are required to reach the nodes on the other processes (ghost nodes). As a consequence,

$$se.hop = \max\ k \quad \text{where} \ \ se.org \in N(k)_{@T_i} \quad (5)$$

2) Precise Hop Computation: Since the CAHC algorithm completely ignores the connectivity relationship between nodes for rapid computation, the number of hops is likely to be underestimated. In the PHC method, we estimate a network topology snapshot of the nodes next to the border more accurately, and for the other nodes we apply the CAHC method to pursue a trade-off between computational cost and accuracy.

Each LP i computes $N(1)_{@T_i}$ for every period ΔT as follows.

$$N(1)_{@T_i} = \{u | D_{u@T_i} \leq R_{max} + V_{max} * (LA_{max} + \Delta T)\} \quad (6)$$

This means that the nodes in $N(1)_{@T_i}$ never connect to the nodes on the other LPs before time $T_i + \Delta T + LA_{max}$. Moreover, we may remove such a node that does not have ghost nodes within $R + 2V_{max} * (LA_{max} + \Delta T)$ distance. $N(2)_{@T_i}$ is calculated as $N_{@T_i} - N(1)_{@T_i}$ where $N_{@T_i}$ is the number of nodes allocated to LP i at time T_i. For $k \leq 3$, we use the same formula as Eq. (4).

C. Estimating Multi-hop Minimum Propagation Delay

In this section we exemplify the calculation of $T_H(k)$ assuming IEEE802.11 DCF. We focus on the delay in the MAC and PHY layers in this section, but if the delay in the other layers is known, we may employ it to enlarge $T_H(k)$. Hereafter, we let T_H^k denote the estimated delay of k-th hop. $T_H(k)$ is described as follows.

$$T_H(k) = \sum_{1 \leq h \leq k} T_H^h \quad (7)$$

In the MAC layer, each frame transmission requires DIFS time. Also, with the RTS/CTS mechanism, additional time for a control frame sequence is required, that

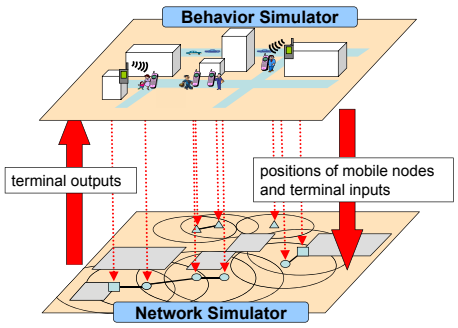


Fig. 3. MobiREAL Simulator

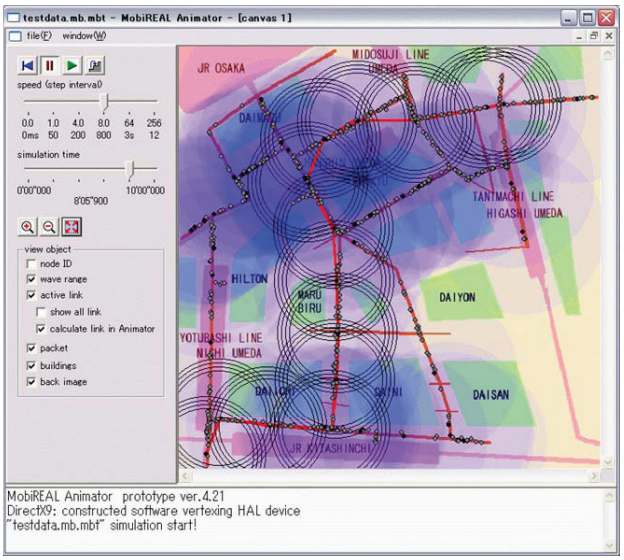


Fig. 4. MobiREAL Animator Snapshot

is, RTS, CTS and ACK transmission delay plus SIFS time between these control frames. In the second or further hops except the last hop, we can no longer expect delay by RTS/CTS even if it was used in the first hop. This is because we do not keep track of unicast packet relay sequences, but focus on data frame sequences which are independent of network layer routing paths. Therefore, for the second and the further hops, we use DIFS time plus transmission delay of minimum size frames. Here, we represent this sequence of nodes as $n_1, ..., n_{k+1}$. In the last hop, the first frame arriving at the ghost node n_{k+1} caused by this frame sequence is CTS frame sent by n_k caused by n_{k-1}'s RTS. Therefore, we consider the SIFS time plus transmission delay as T_H^k. In summary,

$$T_H^1 = \begin{cases} DIFS + \frac{|RTS|+|CTS|+F_{min}+|ACK|}{B} \\ +3 * (SIFS + T_D) & (\textit{RTS/CTS}) \\ DIFS + T_D + \frac{F_{min}}{B} & (\textit{Otherwise}) \end{cases} \quad (8)$$

$$T_H^h = DIFS + T_D + \frac{F_{min}}{B} \quad (2 \leq h \leq k-1) \quad (9)$$

$$T_H^k = SIFS + T_D + \frac{|CTS|}{B} \quad (10)$$

where F_{min}, B and T_D denote the minimum frame size, link capacity and link propagation delay, respectively.

IV. Parallel Simulation in MobiREAL Simulator

We have implemented parallel simulation mechanism as well as the proposed lookahead mechanism into our network simulator MobiREAL [5]–[7].

A. MobiREAL Simulator Overview

The MobiREAL simulator consists of a network simulator and a behavior simulator, which co-work to achieve simulation of real world's node mobility and wireless network systems (Fig. 3). In particular, MobiREAL is original in the point that dynamic behavior of mobiles node can be modeled and simulated easily. We also provide the animator which can visualize packet propagation, routing path as well as the obstacles in the simulated region. Fig. 4 shows a snapshot and additional pictures and movies can be found in [5].

The location and movement of mobile nodes are managed by both network and behavior simulators, but are updated by the behavior simulator only. The updated positions and velocities are notified to the network simulator. Since user behavior is simulated by the behavior simulator, terminal inputs (user decisions) are generated by the behavior simulator and notified to the network simulator. On the other hand, applications are simulated by the network simulator and terminal outputs are delivered to the behavior simulator. For these notification purpose, periodical synchronization of simulation clocks are performed. For k-th synchronization, each simulator processes events with timestamps smaller $k * T$ simulation time (T is the synchronization period), notifies the simulation results (movement of nodes, terminal inputs and outputs) and waits for the other simulator to reach the same simulation time (Fig. 5).

Our network simulator part has been developed based on GTNetS [15], the Georgia Tech. Network Simulator. However, to enable the interaction between the behavior and network simulations, we have implemented several

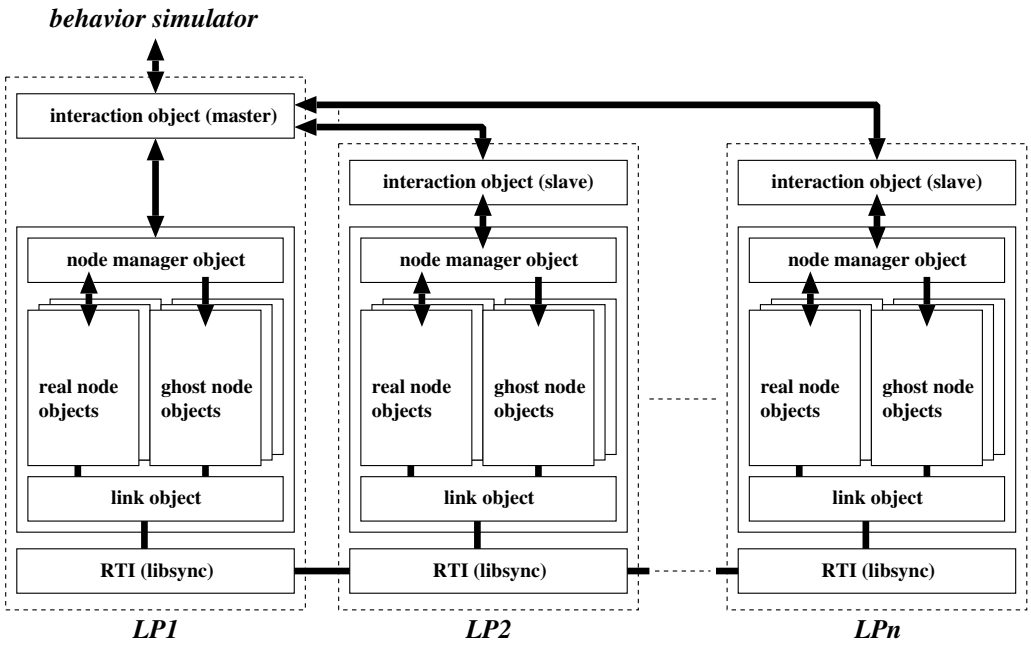


Fig. 6. MobiREAL Parallel Simulator Architecture

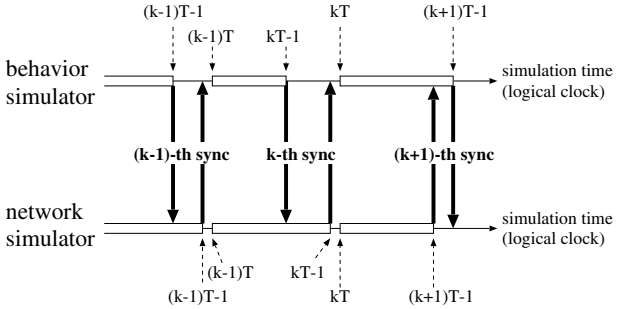


Fig. 5. Synchronization of Two Simulators

components and made modifications on GTNetS. First, we have implemented an interaction class that deals with the interaction with the behavior simulator. Secondly, we have modified the mechanism of GTNetS that deals with node objects so that MobiREAL can manage dynamic generation and removal of node objects. Thirdly, we have introduced signal propagation model under the existence of obstacles. The details of these extensions are described in Ref. [6].

B. Parallel Version of MobiREAL Simulator

The architecture of the parallel version of MobiREAL simulator (called *MobiREAL parallel simulator* hereafter) is depicted in Fig. 6.

GTNetS follows HLA to implement the parallel simulation mechanism. The High Level Architecture (HLA) is a general purpose architecture for distributed computer simulation systems, and is specified as the IEEE standard (IEEE1516). It is intended to hide heterogeneity of various simulators designed for different purposes. The services such as data exchanges and time synchronization which are specified in HLA are implemented as RunTime Infrastructure (RTI). In the GTNetS simulator, they use *libSynk* [16], which has also been developed in Georgia Institute of Technology. The details of GTNetS parallel simulation architecture is given in Ref. [17]. Based on this mechanism, we have implemented several facilities including the methodology presented in this paper. We omit the technical details but the software is now distributed for researchers in academic organizations [5].

V. Experimental Results

The simulation field was 1,000m^2 square. The average number of nodes was 1,200. The Urban Pedestrians Flow (UPF) model and the RWP/ob model [5] [6] were used as the mobility models. We have used IEEE802.11 DCF with RTS/CTS and the DSR routing protocol. The application traffic was CBR over UDP. The simulation time was 300sec., and we have used up to 4 machines of the same specification (Intel Xeon 3.06GHz CPU, 1.5GB Memory). We have prepared three different scenarios, A, B and C shown in Table I.

For the comparison purpose, we have implemented the version where the static lookahead $SIFS + T_D$ is used (referred to as *static version*). Thus this version always use the minimum single hop delay. Also we have

TABLE I
SIMULATION SETTINGS

	Scenario A	Scenario B	Scenario C
Mobility Model	UPF Model	RWP/ob Model	UPF Model
Flow Rates (packets/sec), # of Flows	$10, 4$	$10, 4$	$5, 2$

implemented two versions where the CAHC (cost-aware hop calculation) and the PHC (precise hop calculation) methods are used to compute lookahead. They are referred to as *dynamic-CAHC* method and *dynamic-PHC* method, respectively.

Fig. 7(a) shows the average memory volume per LP. We can see that the increment of memory volume is inverse proportional to the number of LPs, which is good feature for large-scale simulations. Regarding the memory volume, no significant difference among the three versions were found. Then we look at the *performance improvement ratio (PIR)*, which is the ratio of sequential simulation time over the parallel simulation time, in Fig. 7(b). Since the scenario C has less traffic volume than the scenarios A and B, PIR is relatively large. Through all the scenarios, the parallel simulations could dominate the sequential simulations in case of $|LP| = 2$. Also, our dynamic versions could outperform the static version, and in case of $|LP| = 4$, the PIR of static version is less than 1. Since the lookahead in the static version is small, the number of calculations of EIT in the static version becomes large as shown in Fig. 7(c), which greatly affects the speed of simulations. On the other hand, the dynamic versions could improve the speed in all the cases. Compared with the dynamic-PHC, the dynamic CAHC could perform well because of less overhead of hop calculation. We note that in general, the performance improvement is not proportional to the number of LPs in parallel simulations because there is a certain overhead of exchange messages and distributed management of nodes among LPs.

To see the difference of two dynamic versions in more details, we have measured the three metrics, (i) the average lookahead, (ii) time to calculate lookahead, and (iii) the predicted number of hops. The results are shown in Table. II. As expected, the computational cost (calculation time) for PHC is quite larger than CAHC. On the other hand, PHC could predict the number of hops more precisely, and consequently the lookahead is longer than that of CAHC. Clearly there is a certain trade-off, but due to the fact that lookahead improvement of PHC over CAHC is not so large (up to 10%) and PHC required double computational cost of CAHC, in general CAHC is more efficient.

VI. CONCLUSION

In this paper, we have proposed a method to improve the performance of parallel simulation of large-scale multi-hop mobile wireless networks. The idea is to predict the lower bounds of timestamps of potential upcoming events by estimating minimum multi-hop propagation delay. We have implemented the parallel and distributed version of our network simulator MobiREAL, and have also implemented this mechanism. The experimental results have shown that we could achieve 50% speed up in average compared with the version using static lookahead in the simulation of end-to-end communications over 1,200 nodes.

We would like to summarize our contributions. First, we propose a methodology to improve lookahead focusing on *multi-hop propagation delay* in mobile wireless networks. Secondly, to enable the estimation of such multi-hop propagation delay which differs in time and location, we present a technique to *compute lookahead in run-time*. As far as we know, most of the existing methodologies exploit the delay incurred in protocol stacks and/or radio propagation to improve lookahead, and hence we believe that the presented idea is original. We would like to emphasize the fact that this technique can co-exist with the existing techniques that exploit static lookahead. That is, the technique does not compete with those techniques, and thus can be implemented in many parallel discrete event wireless network simulators. Thirdly, we have implemented the mechanism into our network simulator MobiREAL and conducted some experiments.

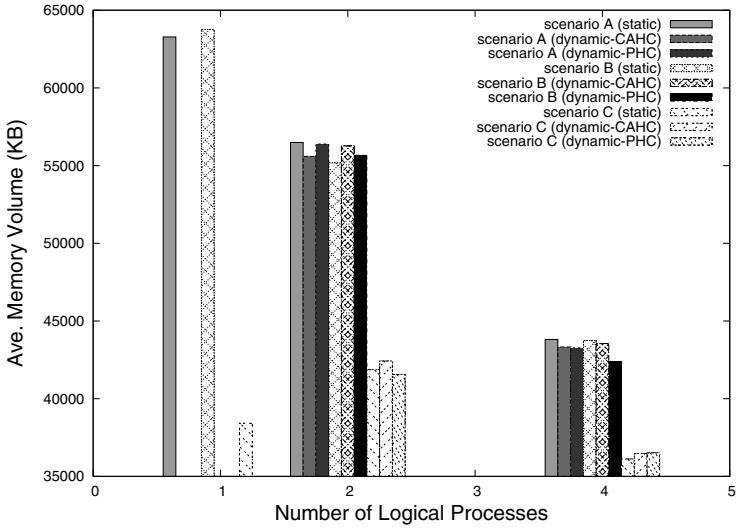
We are going to analyze in mode details the performance of the MobiREAL parallel simulator under various traffic patterns, different routing protocols, different node mobility and geography.

REFERENCES

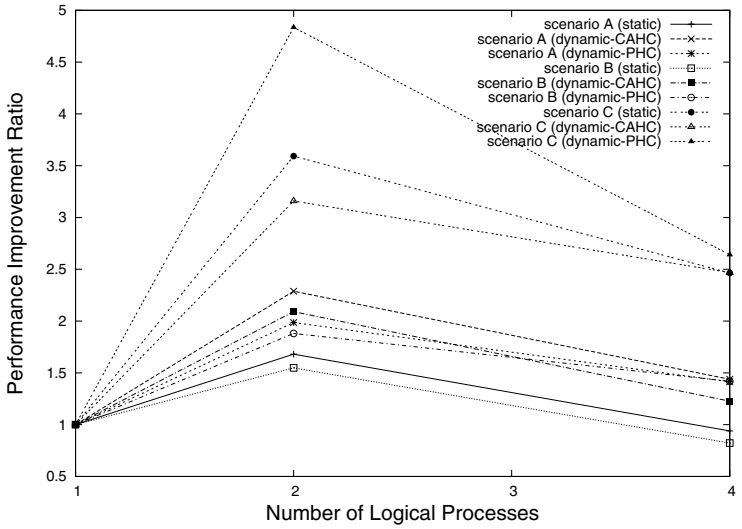
[1] R. M. Fujimoto, "Parallel discrete event simulation," *Communication of the ACM*, vol. 33, no. 10, pp. 30–53, October 1990.

[2] O. Kelly, J. Lai, N. B. Mandayam, A. T. Ogielski, J. Panchal, and R. D. Yates, "Scalable parallel simulations of wireless networks with WiPPET: Modeling of radio propagation, mobility and protocols," *Mobile Networks and Applications*, vol. 5, no. 3, pp. 199–208, 2000.

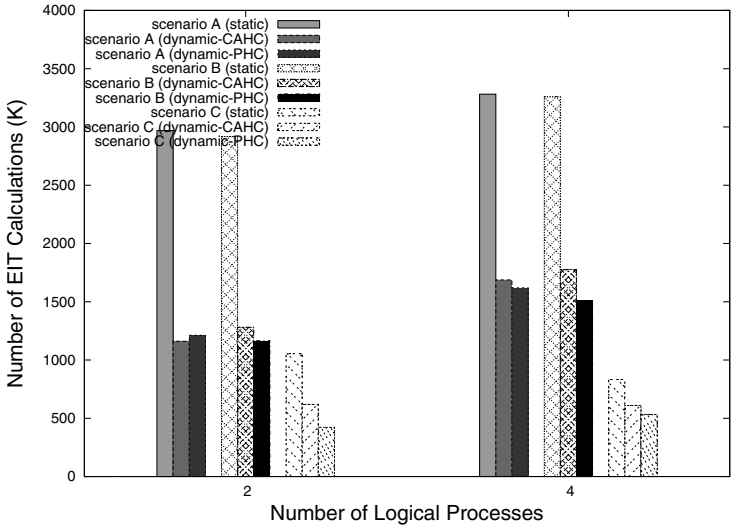
[3] K. M. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Communication of the ACM*, vol. 24, no. 4, pp. 198–206, April 1981.



(a) Average Memory Volume per LP



(b) Performance Improvement Ratio (PIR)



(c) # of Calculation of EIT

Fig. 7. Experimental Results: Parallel Simulation Performance

TABLE II

COMPARISON OF DYNAMIC-CAHC AND DYNAMIC-PHC

		Scenario A		Scenario B		Scenario C	
	$\|LP\|$	CAHC	PHC	CAHC	PHC	CAHC	PHC
Ave. Lookahead (μs)	2	114.2	121.5	115.5	124.2	97.8	107.9
	4	87.3	93.0	88.5	93.8	82.0	86.5
Lookahead Calculation Time (s)	2	69.8	156.6	55.8	117.9	34.7	36.2
	4	32.9	65.6	24.5	42.5	17.3	37.0
Predicted Number of Hops	2	2.5	2.7	2.7	2.9	2.1	2.4
	4	1.9	2.0	1.8	2.0	1.6	1.7

[4] D. R. Jefferson, "Virtual time," *ACM Transactions on Programing Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404–425, July 1985.

[5] "MobiREAL," http://www.mobireal.net.

[6] K. Konishi, K. Maeda, K. Sato, A. Yamasaki, H. Yamaguchi, K. Yasumoto, and T. Higashino, "MobiREAL simulator - evaluating MANET applications in real environments," in *Proc. of the IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS2005)*, 2005, pp. 499 – 502.

[7] K. Maeda, K. Sato, K. Konishi, A. Yamasaki, A. Uchiyama, H. Yamaguchi, K. Yasumoto, and T. Higashino, "Getting urban pedestrian flow from simple observation: Realistic mobility generation in wireless network simulation," in *Proc. of the 8th ACM/IEEE Int. Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM2005)*, 2005, pp. 151 – 158.

[8] J. Panchal, O. Kelly, J. Lai, N. Mandayam, A. T. Ogielski, and R. Yates, "WiPPET, a virtual testbed for parallel simulations of wireless networks," in *Proc. of the 12th workshop on Parallel and Distributed Simulation (PADS'98)*, 1998, pp. 162 – 169.

[9] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette, "GTW: a time warp system for shared memory multiprocessors," in *1994 Winter Simulation Conference*, 1994, pp. 1332–1339.

[10] A. Boukerche and A. Fabbri, "Partitioning parallel simulation of wireless network," in *Proc. of the 2000 Winter Simulation Conference*, 2000, pp. 1449 – 1457.

[11] Z. Ji, J. Zhou, M. Takai, J. Martin, and R. Bagrodia, "Optimizing parallel execution of detailed wireless network simulation," in *Proc. of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)*, 2004, pp. 162 – 169.

[12] J. Liu and D. Nicol, "Lookahead revisited in wireless network simulations," in *Proc. of the 16th Workshop on Parallel and Distributed Simulation(PADS'02)*, 2002.

[13] R. A. Meyer and R. L. Bagrodia, "Improving lookahead in parallel wireless network simulation," in *Proc. of the 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'98)*, 1998, pp. 262–267.

[14] "Qualnet," http://www.scalable-networks.com/.

[15] G. F. Riley, "The Georgia Tech network simulator," in *Proc. of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, 2003, pp. 5 – 12.

[16] "libSynk," http://www.cc.gatech.edu/fac/kalyan/libsynk.htm.

[17] G. F. Riley, T. M. Jaafar, R. M. Fujimoto, and M. H. Ammar, "Space-parallel network simulation using ghosts," in *Proc. of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)*, 2004, pp. 171–177.