International Journal of Future Generation Communication and Networking
Vol. 9, No. 4 (2016), pp.301-314
http://dx.doi.org/10.14257/ijfgcn.2016.9.4.27

TimSim: An Efficient Simulator for Wireless Ad-hoc Networks

Hui Xia, Jia Yu, Guo-Dong Wang and Zhen-Kuan Pan

College of Computer Science and Technology, Qingdao University, Qingdao 266071, P.R.C
Shandong Provincial Key Laboratory of Software Engineering, Shandong University, Ji'nan 250101, P.R.C
Postdoctoral Research Station of System Science, College of Automation and Electrical Engineering, Qingdao University, Qingdao 266071, P.R.C
Corresponding E-mail: *xiahui@sdu.edu.cn*

Abstract

With the growth of increasingly complex in wireless ad-hoc networks, performance modeling and evaluation play a crucial part in their design process to ensure successful deployment and exploitation in practice. Compared with the actual implementation (testbeds), simulation has become a valuable tool, which allows studying large scale systems that cannot be built practically. However, most simulators currently used for such networks are intended to evaluate individual routing protocol, which should be extracted from the source code and rewritten in accordance with the simulator's requirements. Thus, in this paper, we design a new wireless ad-hoc network simulator, called Time Step-based Wireless Ad-hoc Network Simulator (TimSim). It is intended to facilitate the migration of simulation code to real devices via providing useful APIs with real device driver. We use a discrete event processor with time step-based feature as the simulating engine, and the data can be transmitted in bit-level. A unique feature of TimSim is its ability to support the simulation of multi-thread programming. Additionally, we import database access to store the simulation information. Extensive experiments have been conducted to evaluate the efficiency of TimSim. Furthermore, a case study on a real system has been presented to illustrate its practicability.

Keywords: *Wireless Ad-Hoc Networks, Simulator, Discrete Event Processor, Time Step-Based, Multi-Thread, Database*

1. Introduction

A primary aim of wireless ad-hoc networks is to deliver data in areas where there is no pre-defined infrastructure. These networks have recently witnessed their fastest growth period ever in history. Real wireless ad-hoc networks are now implemented, deployed and tested, and this trend is likely to increase in the future. However, as such networks are increasingly complex, performance modeling and evaluation play a crucial part in their design process to ensure their successful deployment and exploitation in practice. Moreover, efficiently deploying and monitoring a large number of hosts in such networks involve high cost and intensive labor.

There are mainly two basic approaches to investigate wireless ad hoc networks, as experimentation (testbeds) and software-based simulation. Experimentation (testbeds) provides a more realistic test environment. Experimentation allows users to get more accurate test results and important details that might be missed by simulation. However, building testbeds and managing a large number of wireless devices involve high cost and intensive labor. In addition, because of the environment in a testbed may change dramatically, getting reproducible results is difficult [1]. Oppositely, software-based

ISSN: 2233-7857 IJFGCN

simulation turns out to be an applicable and useful means. Simulators can handle the network as a whole, thus managing hosts in the network becomes easier. By artificially controlling the environment conditions, simulation provides excellent reproducibility across experimental trials. Network simulators are built to allow the study and evaluation of protocols and applications developed by researchers. Advantages of a common simulator are described in [2].

By looking at the current status of wireless ad-hoc network simulators, a small amount of simulators facilitate the migration of the simulation code to real devices [3]. Most of them focus on evaluating individual routing protocols or algorithms, while simulating the application's source code directly is not supported. When simulating an application of a new designed protocol, we have to extract this protocol from the source code and rewrite it in accordance with the simulator's requirements. In addition, any of these simulators can hardly store all the detailed data generated during the simulation. Most of them just generate a trace and statistics file after simulation.

To address these above questions, in this paper, we introduce a new network simulator TimSim (Time Step-based Wireless Network Simulator). The motivations for designing this new simulator are: (a) This new simulator would be dependable, valuable and easy to use; (b) It can provide a scalable and useful platform to help researchers simulate and analyze their wireless network applications; (c) It can also facilitate the migration of the simulation code to real devices.

The main technical contributions of our work are summarized as follows: (1) Providing common meaningful APIs with device driver. Using these APIs, users can simulate their source code with only a few modifications; (2) Driven by a time step-based simulation engine. Using a time step-based discrete event model, TimSim can get simulation data in detail under a good performance. More precisely, TimSim can simulate the data transmitted over wireless ad-hoc networks in bit-level; (3) Supporting multi-thread programming techniques. The core of this measure is using a simulation engine thread drive multiple simulation nodes threads. Threads can achieve synchronization via using the shared variables and setting up the events; (4) Storing all the simulation data into a database. After simulation, users can analyze the behavior of a simulating host in detail; (5) Designing a friendly GUI, which is used to provide an efficient front-end. With the friendly graphical interface, users can easily analyze the simulation results and have an intuitive understanding of a simulation.

The remaining paper is organized as follows. Section 2 starts with a discussion of related works. In Section 3, we propose the design and detail features of TimSim simulator. Section 4 presents the simulation flow of TimSim and describes how to use TimSim. The performance evaluation and two test cases are presented in Section 5. Finally, section 6 gives the concluding remarks along with extensions and directions for future work.

2. Related Work

Some simulators [4] have been proposed after the wireless extension of NS2 [5], which considered as the first ad-hoc networks simulator. Hogie and Bouvry [3] described less than twenty ad-hoc networks simulators currently in use. In the following text, we will discuss four most widely used simulators.

NS2 [5] is a discrete-event simulator targeted at networking research. NS2 is now the most widely used simulator in MANET research for its open source and high accurate PHY. Researchers can use NS2 by adding their own C++ module which implements the network routing protocols and corresponding OTCL scripts which is used for simulation network configuration [6, 7]. NS2 is now maintained by NS2 community and volunteers from all over the world. Some novel approaches have been proposed to improve NS2's performance. For example, Walsh and Sirer [8] proposed staged simulation to improve run time performance and scale of discrete event simulators and applying the staged

simulation to NS2. However, lack of modularity, inherent complexity and high consumption of computational resources are its typical disadvantages.

OPNET [9] is a discrete-event network simulator first proposed by MIT in 1986. It is now the most widely used commercial simulator. OPNET provides a comprehensive development environment for the simulation and performance analysis of communication networks. Particularly, it can execute and monitor several scenarios in a concurrent manner. As a commercial product, OPNET has a well engineered user-interface using mainstream software and operating system. However, compared to the freeware nature of NS-2, there is a need to enter into an OPNET Modeler license agreement and associated direct costs [10, 11].

OMNET++ is a powerful object-oriented modular discrete event simulator [12] written in C++. OMNET++ was developed to be a powerful open-source discrete event simulation tool that can be used by academic, educational and research-oriented commercial institutions for the simulation of computer networks [13]. OMNET++ can facilitate visualizing and debugging of simulation models so that it can reduce debugging time. Additionally, OMNET++ provides an integrated development environment and open data interfaces. However, when using OMNET++ to simulate our protocols, we have to rewrite the code in order to migrate it to real devices.

GloMoSim [14, 15] is a library-based sequential and parallelized simulator for wireless networks. It is developed at UCLA. It is developed using PARSEC, a C-based parallel simulation language, so it can run on SMP (shared-memory symmetric processors) computers. Using GloMoSim will get accurate performance prediction of large-scale network models using parallel execution on a diverse set of parallel computers. However, using GloMoSim involve writing new protocols and modules in PARSEC, will be unfamiliar to most of the current used simulators for the same time.

3. Design of Timsim

Building with a module approach, TimSim is decomposed into three layers and each layer has its corresponding modules. Figure 1 shows the architecture of TimSim. In the following section, we will describe each layer in detail.

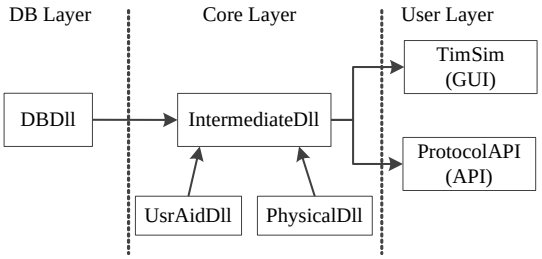

Figure 1. Timsim Architecture

3.1 User Layer

User Layer provides all the necessary interfaces for researchers to use. There are two modules in User Layer. TimSim (GUI) is the graphic user interface (GUI) module, which implements the function of showing the necessary simulation information for researchers; Protocol API provides useful APIs for researchers to use. Particularly, these APIs have the same prototype as the real device drivers provided, so that protocols or applications can directly migrate to the real device from simulator.

TimSim (GUI) module supports the graphical representation of the simulation. It provides a friendly front-end to researchers using TimSim. Figure 2 shows the main GUI of TimSim. There are four viewing areas in the main window. Network Element Tree View Area lists all the elements (nodes, interferences and obstacles) in the simulation

network. Network elements are organized in a Tree-View. Network Element Graphic View Area represents the simulation network elements in a graphical way. In TimSim we use different color presents different node's state. Once simulation starts, any change of nodes' state will be shown in this area, thus we can have an intuitively understanding of the simulation. Network Element Properties View Area shows the properties of the selected element in detail. For instance, for a selected node <the node's ID, positions, state and other physical characteristics>, will be shown in a detail way. Network Output View Area prints all the simulation logs, such as simulation time, transmission data, and nodes state etc.

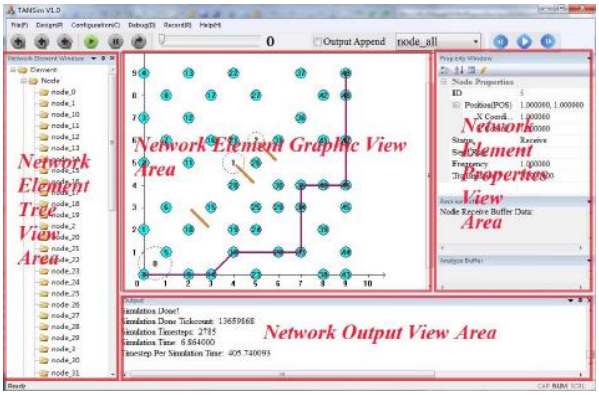

Figure 2. Timsim Main GUI

ProtocolAPI (API) module allows researcher's program to directly migrate from simulation environment to real device nodes, by providing lots of useful Application Programming Interfaces (APIs).

Table 1. System Level Apis

Name	*Description*
SystemSleepSimAPI	API for system sleep function
SystemCreateThreadSimAPI	API for system create thread function

Table 2. Wireless Node Level Apis

Name	*Description*
NodeSetStatusSim	Set node's status API
NodeSetFreqSim	Set node's frenquency API
NodeSetSendPowerSim	Set node's send power API
NodeForceTerminateSim	Force terminate node API
NodeCSMASim	Node carrier sense multiple access API
NodeSendSim	Node block send API
NodeReceiveSim	Node receive API
NodeInitReceiveBufSim	Init node receive buffer API
NodeGetReceiveBuflenSim	Get node receive buffer's length API
NodeDelReceiveBufSim	Delete data in node receive buffer API

There are two main categories APIs that TimSim provided: *System Level APIs* (shown in Table 1) provides the APIs in system level. Sleep and CreateThread are the most widely used interfaces in protocols and applications in ad-hoc network systems. So we firstly provide these two APIs, and other system level APIs will be provided in the future work. *Wireless Node Level APIs* (shown in Table 2) provides the same necessary functions as real wireless device may provide. Researchers can invoke these APIs just like invoke those functions provided by real device drivers.

3.2 Core Layer

Core Layer is the simulation core of TimSim. There are two assistant sub-modules and one main module in the core layer. *IntermediateDll* module, using a discrete-event processor with time step-based feature, is the simulating engine of TimSim. *UsrAidDll* sub-module provides some useful low-level functions for the *IntermediateDll* module; *PhysicalDll* sub-module realizes the simulation of radio propagation model.

UsrAidDll module is a functional sub-module. *UsrAidDll* defines some useful data types and provides a lot of low-level data structure operation, such as ring buffer memory management operations, and linked list operations etc. These low-level functions will be invoked frequently. Separate this sub-module from other modules improves code reusability. So any module wants to invoke these functions just load the DLL module and does not need to write those functions again. This can also give us help when we want to modify those functions, as we only need to modify the code in *UsrAidDll*.

PhysicalDll module implements the physical layer simulation of ad-hoc network. The simulating model's accuracy is important in a simulator. However, none of network simulators is accurate, only the testbeds can be said to be accurate. Thus, at best, we can only say the simulator is dependable and realistic. So what we can do is to provide a more realistic propagation model. In order to get the relatively realistic simulation result, we propose an enhanced free path loss propagation model [16], without mobility patterns considered. Kathirvel and Srinivasan [16] considered path loss and shadowing in their propagation model. We will focus on the error bit analysis derived from the typical model.

Enhanced Error bit analysis model: In *PhysicalDll* module we propose an enhanced error bit analysis model based on calculated bit error rate (*BER*) from the free path loss propagation model.

In this enhanced propagation model, we can get bit-level data of each packet, including the error bits. For every stage with a stable Signal to Noise Ratio (SNR), we can simulate the received data in the following steps:

(*a*) Firstly, we calculate the BER (Bit Error Rate) according to the typical free path loss propagation model using SNR as an input argument. With the obtained BER, we can get the probability of *k* error bits in a particular stage from the Equation (1):

$$P_k = p^k (1-p)^{N-k} C_N^k \tag{1}$$

Where *N* presents the total number bits in current stable stage, presents *k* error bits. And *p* presents current *BER*, which means the probability of a bit to be error.

(*b*) Then, we use the following quote Equation (2) to get the real error bit numbers. At first, we set *R* to a random value between 0 and 1. By initiating *k* to 1, and constantly increase it by one until reaches *N*. In the processing, if we find the non-equality satisfies, we allocate *i* error bits.

$$\sum_{k=0}^{i} P_k > R \tag{2}$$

(*c*) After we get the number of error bits, we set the real error bits. The error bits setting process is composed of two stages: (1) Find the real error bit randomly; (2) Set the error effect randomly, which may be just discarding the bit, or overturn the bit (0->1, or 1->0). Using the enhanced error bit analysis model, we can get the detailed received data, with consideration of the network environment. With these data researchers can know what exactly happened in simulation, not just some statistics data. What's more, TimSim can simulate the coincidence that node can receive correct checksum but wrong data. And two or more errors may be cancelled out. This also improves the reality degree of simulation compared to other simulators.

IntermediateDll module is the simulating core of TimSim. It implements a discrete-event simulation engine with time step-based feature. Network Simulation can be

either continuous or discrete. Generally speaking, continuous simulation makes use of analytic models to simulate networks. Considering the intrinsic complexity and dynamic nature of ad-hoc networks, analytic models can hardly be applied. Thus discrete simulation turns out to be a more practicable approach. TimSim inherits the core feature of currently used discrete-event simulators, and builds up with new time step-based features. In *IntermediateDll* module, we proposed two novel models to implement the simulation engine.

Time step-based Discrete-event model: (*a*) Firstly, we define a global simulation time step variable-**GST**. GST is initialized to be zero and will be increased step by step; (*b*) Then we describe the event happens in simulation. When simulation starts, every simulation device node has its initial state. Most frequently, a source node want to send some data to a special target node. Through our propagation model, we can calculate when and how the data to be sent and received. We call such node send or receive behavior as an event happens in the simulation network; (*c*) After have a better known of the two above definition, we describe how the time step-based discrete-event model works. For a simulation node when there is an event will happen in the future through current calculation, its flag will be set. TimSim will check the flag state of every node when GTS reaches to a new value. If the flag of a node is set, TimSim will handle the corresponding event. Otherwise, if all nodes' flags are not set TimSim will simply increase the GTS and do nothing, which can be done very fast on current mainstream computers.

When simulation starts, every node has its initial status. In most cases, a source node wants to send some data to a special target node. When a node begins to send some data, nodes that can receive the data can be calculated through our propagation model. For these nodes, their flags will be set to identify there are events to handle. When GST reaches to a new value, TimSim will check every node's flag to see whether having events to handle. If not, TimSim will simply increase the GST. Realizing with bitmap, this checking and increasing process can be done very fast on current mainstream computers. As a result, when the network is much quiet, GST can increase quickly. Nevertheless, if there are lots of data transmission in the network, TimSim will spend much execution time on handling these events, and then GST will increase slowly.

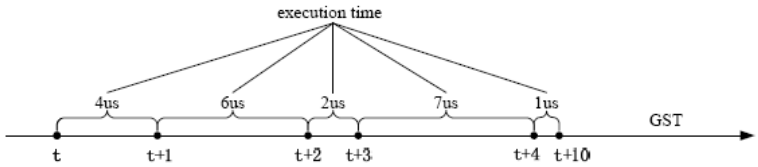

Figure 3. Time Step-Based Discrete Event Model

As a visual description, Figure 3 shows the effect of this model. We can see that the execution time spending on the same GST interval is different. For instance, increasing GST from t+3 to t+4 uses 7*us*, but increasing it from t+4 to t+10 only uses 1*us*.

In most cases, we define the minimum time of sending or receiving a bit data as one time step. As a result, we can get the data transmitted over wireless ad-hoc networks in bit-level. Since TimSim stored all the bit-level data of each packet, it needs more storage space compared with other simulator's trace file. We are actively investigating methods to optimize the storage space using some compression techniques. This will be improved in future versions.

Using this time step increasing mechanism, TimSim only need to set the flag and don't have to maintain an event queue. Maintain such event queue may require lots of resources when the simulation network becomes very large. Furthermore, using GTS can provide synchronization for all the nodes' threads, which is very important in supporting

multi-thread programming and we will talk it in more detail in the following part.

Multi-thread programming model: TimSim uses multi-thread programming model in system realization. There is a main simulation thread in charge of the core simulating and a GUI thread focusing on the graphic representation. More important, there are other threads to support the simulation of multi-thread programming [17].

As a unique character, TimSim supports the simulation of multi-thread programming. First, TimSim al-locates a thread for each simulation node, thus each node can run its own application. Then TimSim provides thread creating API, which allows the application run over simulation node to be multi-thread, application can create thread using the API just like creating thread in a real device node. This also contributes to migrating code from simulator to real device more freely.

We use critical section and setting and waiting event operations to manage the threads. And use the global GTS to implement the synchronization of different threads.

3.3 DB Layer

DB Layer is in charge of database access control. TimSim stores the simulation data into the database when simulating and fetches the data from the database when re-showing the process of simulation. After giving an overview of each layer and its modules, we will describe each module of TimSim in detail.

DBDll Module

TimSim stores the whole simulation data into database so that researchers can review the simulation in any time. Using the stored simulation data, developers can analyze the protocols or applications in an easy and intuitive way.

DBDll module implements the database access function. Following tables are designed in a relation database system.

node_info table: Storing all the simulation nodes' initial configurations.
inoise_info table: Storing all the simulation interferences' configurations.
obstacle_info table: Storing all the simulation obstacles' configurations.
design_info table: Storing the design information of each simulation.
debug_info table: Storing all logs and debug information generated in simulation.
node_time_info table: Storing all nodes' information in a certain time.

Firstly, TimSim will store the simulation design and all the simulation network elements' initial configurations to the corresponding tables.

Then, once the simulation starts, every change of each node will be stored into the ***node_time_info*** table and every output in the Network Output View Area will be stored into the ***debug_info*** table.

After simulation, researchers can choose a particular simulation design stored in the database then load it. After that researchers can reshow the simulation in a very detail way. For instance, we can only focus on a particular node, or see what happened at a particular time, via selecting it from the ComboBox in the toolbar. Storing the simulating information into the database rather than a trace file, can help a lot for users to analyze their simulation. More significantly, users can get extra information, like frame loss rate, by writing additional code in their simulation design.

4. Simulation Flow and How to Use Timsim

In this section, we present the simulation flow of our new simulator and describe how to use it.

4.1 Simulation Flow of Timsim

The simulation flow of TimSim is designed to be effective with high scalability and easy to understand. Figure 4 shows the detailed simulation flow of TimSim.

After loading and initializing all of the simulation network elements, users can press the simulate button. TimSim will first initialize the global critical section used for sharing data in different threads. Then TimSim will allocate thread parameters and create the thread for each node. Concurrently, it will create the simulating engine thread and initialize the simulating parameters.

Then protocols or applications will be run on each node's thread. Simulating engine thread will drive the nodes' thread to process. All these threads can communicate with each other through sharing data and Message Passing Interface (MPI). Furthermore, these threads realize synchronization using the multi-thread management we have described in the above section.

After the simulation process is finished, the simulating engine thread will wait all nodes threads to be terminated. And main thread of TimSim will wait the simulating engine to be done. Then after all the child threads terminated, TimSim finally free the allocated thread parameters and delete the global critical section.

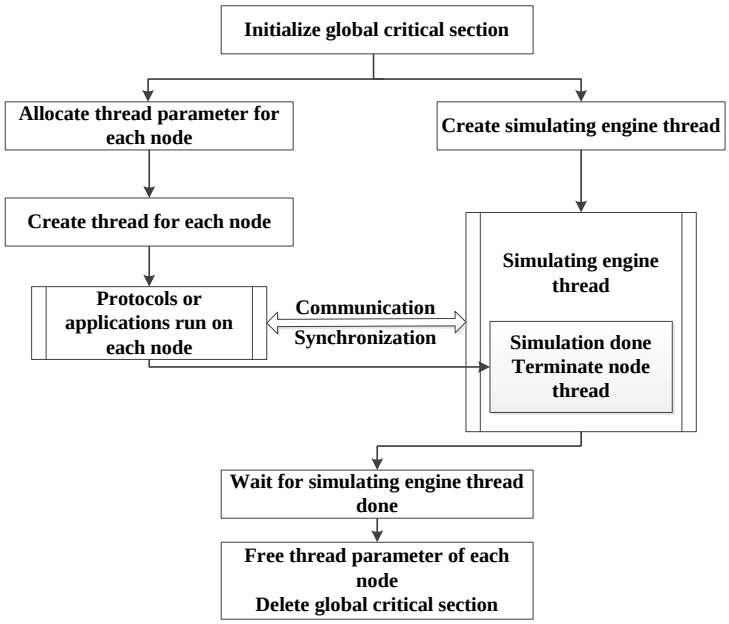

Figure 4. Simulation Flow of Timsim

4.2 Using TimSim

Users can use TimSim in the following steps:

1) Write your protocol or application code using the APIs provided by ProtocolAPI module. Additionally, users must load the ProtocolAPI module and include the header file in which contains the declaration of those APIs.

2) Configure the environment of simulation network. Users can set the simulation network element through the GUI or just loading the configuration INI file.

3) Run the simulation. When the simulation is running, simulation data will be stored into the database and log information will be printed in the Network Output View Area automatically. When the simulation is done, TimSim will inform users with a message dialog.

4) Analyze the simulation result. Users can analyze the simulation results through checking the printed log information after simulation done. Additionally, users can also re-see the simulation using the saved information in the database.

5. Performance Evaluation and Case Study

In this section we firstly evaluated the performance of TimSim and described test cases to verify the practicability of this new simulator. In this subsection, all the experiments are carried out on a PC machine with a 2GB installed RAM and 2.27GHz Intel duo-core processor. In order to verify the performance of TimSim, we ran some experiments and monitored the time-steps per simulate time and the memory usage as the number of node increases.

5.1 Performance Evaluation

Figure 5 shows that the ratio (time-steps per simulation time) varies with the increase of numbers of nodes. In the experiment we set three different simulation settings. As shown in Figure 5: (1) Necessary Debug Info means that TimSim will only print and store the necessary debug information when simulation; (2) All Debug Info Print means TimSim will print all the debug information besides the Necessary Debug Info; (3) All Debug Info Print & Store will print and store all the debug information in addition to the All Debug Info Print. As expected, time-steps per simulation time decreases with respect to the number of nodes increases. That means TimSim will cost much more time to simulate unit real time. There are two factors included in this condition, the first is the simulation time TimSim used to simulate and the second is the time cost in the real network. At start the ratio is high, and it won't change much when the number of nodes increases to big enough. The reason is there is a balance between those two factors. Additionally, from the three simulation settings we find that the database operation will cost the most simulation time. So if users don't care much about the details, they can just use the simple setting.

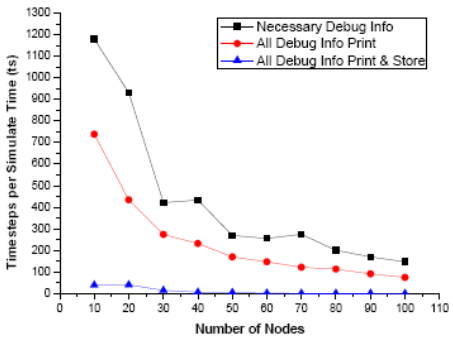

Figure 5. Time Steps

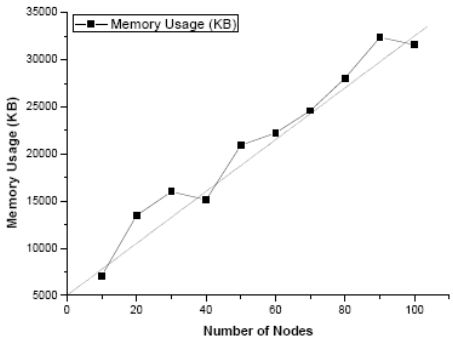

Figure 6. Memory Usage

From Figure 6, we can see that the memory usage of TimSim increases linearly with an acceptable quantity as the number of nodes increases. Due to using the object-oriented programming method, we see very node as an instance of the node object and allocate memory for it. Significantly, TimSim will copy the data packet from the sender to every receiver. With these two reasons memory usage increases with respect to the number of nodes increase.

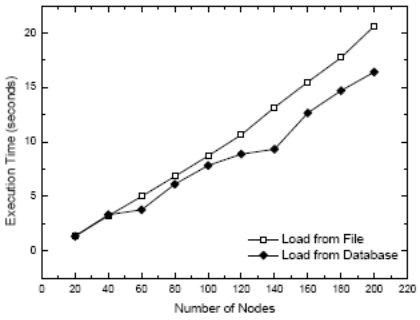

Figure 7. Execution Time with Two Different Configurations

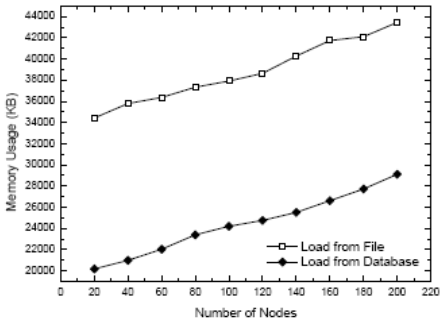

Figure 8. Memory Usage with Two Different Configurations

The simulating execution time results are plotted in Figure 7. As expected, execution time scales linearly with the number of nodes. For a stationary wireless ad-hoc network, TimSim will calculate the potential reachable nodes at the very beginning and cache them. Thus every node just checks its cached nodes before every sending operation and do not have to check each node in the network. This improves the computation complexity from quadratic growth to linear growth. Besides, loading configuration from database can save some time due to the high efficiency of database access compared with the low efficiency of configuration file operation.

The memory usage results with different ways of storage are plotted in Figure 8. Memory usage increases linearly with the number of nodes. Meanwhile TimSim has an acceptable quantity of memory usage. Due to object-oriented realization, TimSim allocates fixed memory for each node object. Thus TimSim's memory consumption increases linearly with the number of simulating nodes. When loading configuration from file, TimSim have to allocate additional memory to process the configuration file. As a result, loading configuration from file will consume more memory.

Figure 9 shows the execution time as the simulating network density increases. As expected, increasing the network density caused a linear increasing of the execution time. When network density increases, the network traffic also increases, finally causing the execution time increases. And the relationship between them is linear. This can be explained by a very simple example. A node wants to send some data to its neighbors. If the network density is 5, which means that a node have 5 neighbors, then TimSim will have to handle the receiving events of these 5 neighbors. When the density increases to 10, TimSim have to handle all the 10 neighbors' receiving events. As a result, the execution time has a linear relationship with the network density.

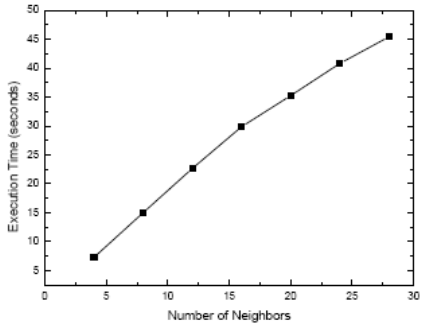

Figure 9. Execution Time with Increased Network Density

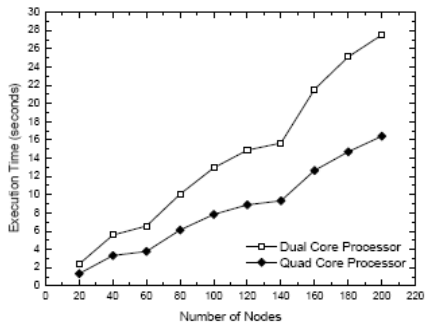

Figure 10. Execution Time Comparison with Different Processors

Figure 10 shows the execution time simulated on two different CPU (a new configuration, PC machine with a 4GB installed RAM, 2.27GHz Intel dual-core processor). As expected, simulation on a quad-core processor is faster than on a dual-core processor. Due to the multi-thread programming realization, TimSim can get better performance when the processor has more cores.

5.2 An Application with TimSim

In this part we describe an application [18] to illustrate the practicability of TimSim. In the above subsection, we have applied TimSim to the route protocol design on a wireless meter reading system.

In real wireless networks, CSMA (Carrier Sense multiple Access) is a very valuable technique to avoid potential collisions in communications, and carrier sensing time is an important impact factor. In our test case, carrier sensing time is set by CSMA BASE TIME added by a random integer CSMA RAND RANGE. We set 50 wireless device nodes, 3 interfaces, and 3 obstacles in this test case.

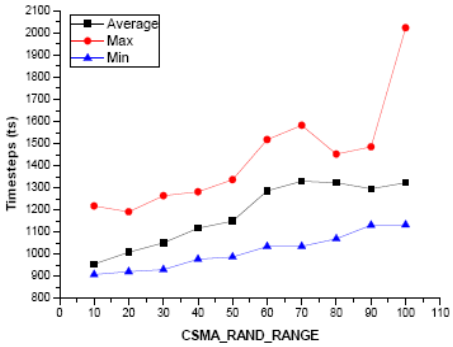

Figure 11. Time Steps Cost

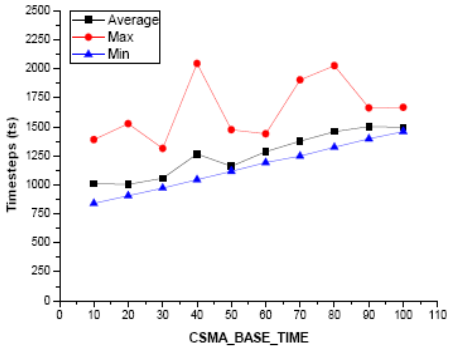

Figure 12. Time Steps Cost

Figure 11 and Figure 12 shows the time steps spent on route discovery as the CSMA_BASE_TIME and the CSMA_RAND_RANGE increases. In our test case, we do 10 simulations for every CSMA_BASE_TIME and CSMA_RAND_RANGE value, and give the maximum, minimum and the average time steps cost. As expected the average and minimum time steps gets higher as the CSMA_BASE_TIME and CSMA_RAND_RANGE increases. However there is a dithering for the maximum time step, because the collision is unpredictable and the maximum time step can vary differently.

6. Conclusions and Future Work

In this paper we have described the design, performance evaluation and validation of TimSim. TimSim uses a new time step-based discrete-event driven simulating technique. Researchers can write their code more efficient by using the common APIs provided by TimSim, and do not have to rewrite their code when migrating from the simulator to the real device nodes. Using the obtained simulation data in database, developers can review the simulation and further analyze the protocols and applications in a more detailed way.

From the performance evaluation and the case study described in the section V, TimSim simulator turns out to be a valuable and efficient simulator for wireless ad-hoc networks. This new simulator will help us a lot when we designing our proposed protocols, especially when we have to set some important factors, such as carrier sense time in CSMA.

We would continue our future work in the two directions: 1. Provide more useful APIs

for developers, so TimSim can be more widely used; 2. Optimize the storage space and access speed of database, so it will be more efficient; 3. Realize the mobility simulation model to support mobile network simulations.

Acknowledgments

We would like to thank anonymous referees for their helpful suggestions to improve this paper.

This research is sponsored by the Natural Science Foundation of China (NSFC) under Grant Nos. 61402245, 61572267 and 61272425, the Project funded by China Postdoctoral Science Foundation under Grand No. 2015T80696 and 2014M551870, the Shandong Provincial Natural Science Foundation No. ZR2014FQ010, the Qingdao Postdoctoral Application Research Funded Project, the Open Project Foundation of Shandong Provincial Key Laboratory of Software Engineering under Grant No. 2013SE01, PAPD, CICAEET and Foundation of Huawei under Grant No.YB2013120027.

References

[1] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordstrom and C. Tschudin, "A large-scale test bed for reproducible ad hoc protocol evaluations", In Wireless Communications and Networking Conference (WCNC2002), **(2002)**, pp. 412-418.

[2] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan and Y. Xu, "Advances in network simulation", Computer, vol. 33, no. 5, **(2000)**, pp. 59–67.

[3] L. Hogie, P. Bouvry and F. Guinand, "An overview of manets simulation. Electronic notes in theoretical computer science", vol. 150, no. 1, **(2006)**, pp. 81-101.

[4] C. Yan, Z. P. Jia, L. Ju, H. Xia and H. Q. Xu, "TimSim: A Timestep-based Wireless Ad-hoc Networks Simulator", In the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom2013), **(2013)**, pp. 1741-1746.

[5] http://www.isi.edu/nsnam/ns/, accessed December, 2012.

[6] T. Issariyakul and E. Hossain, "Introduction to network simulator NS2", Springer Verlag, **(2011)**.

[7] J. H. Chen, T. C. Lien and H. M. Yang, "The implementation of IEEE 802.16m protocol module for ns-3 simulator", Simulation Modelling Practice and Theory, vol. 49, **(2014)**, pp. 41-56.

[8] K. Walsh and E. G. Sirer, "Staged simulation: A general technique for improving simulation scale and performance", ACM Transactions on Modeling and Computer Simulation (TOMACS), vol. 14, no. 2, **(2004)**, pp. 170-195.

[9] F. Desbrandes, S. Bertolotti and L. Dunand, "Opnet 2.4: an environment for communication network modeling and simulation", In Proceedings of the European Simulation Symposium, **(1993)**, pp. 609-614.

[10] G. F. Lucio, M. Paredes-Farrera, E. Jammeh, M. Fleury and M. J. Reed, "Opnet modeler and ns-2: comparing the accuracy of network simulators for packet-level analysis using a network testbed", WSEAS Transactions on Computers, vol. 2, no. 3, **(2003)**, pp. 700-707.

[11] Y. Zaki, T. Weerawardane, C. Görg and T. Giel, "A. Long Term Evolution (LTE) model development within OPNET simulation environment", In OPNET workshop, **(2011)**.

[12] K. Wehrle, M. Günes and J. Gross, "Modeling and Tools for Network Simulation (Eds)", Omnet++, **(2010)**, pp. 35-59.

[13] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment, In Proceedings of the 1st international conference on Simulation tools and techniques for communications", Networks and systems & workshops, vol. 60, **(2008)**.

[14] X. Zeng, R. Bagrodia and M. Gerla, "Glomosim: a library for parallel simulation of large-scale wireless networks", In Parallel and Distributed Simulation, 1998 (PADS 98), **(1998)**, pp. 154-161.

[15] A. Kathirvel, "Introduction to GloMoSim [J]", **(2011)**.

[16] A. Kathirvel and R. Srinivasan, Analysis of propagation model using mobile ad hoc network routing protocols", International Journal of Research and Reviews in Computer Science, vol. 1, no. 1, **(2010)**.

[17] P. J. Ren, M. Lis, M. Cho, K. S. Shim, C. W. Fletcher, O. Khan, N. N. Zheng, S. Devadas, "HORNET: A Cycle-Level Multicore Simulator. IEEE Transactions on Computer-aided", Design of Integrated Circuits and Systems, vol. 31, no.6, **(2012)**, pp. 890-903.

[18] J. L. Yang, Z. W. Li, Y. C. Cai, Q. Zhou, "PowerRush: An Efficient Simulator for Static Power Grid Analysis", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 10, **(2014)**, pp. 2103-2116.

Authors

Hui Xia, born in 1986, is currently a Lecture in the College of Computer Science and Technology at Qingdao University, China. His research interests focus on network and information security, trust computing, mobile computing, embedded system and cryptology. His research is sponsored by the Natural Science Foundation of China (NSFC) under Grant No. 61402245, the Project funded by China Postdoctoral Science Foundation under Grand Nos. 2015T80696 and 2014M551870, the Shandong Provincial Natural Science FoundationNo. ZR2014FQ010, the Qingdao Postdoctoral Application Research Funded Project, the Open Project Foundation of Shandong Provincial Key Laboratory of Software Engineering under Grant No. 2013SE01. (*xiahui@sdu.edu.cn*)

Jia Yu, born in 1976, is currently a Professor in the College of Information Engineering at Qingdao University, China. His main research interests include cryptology, network security, cloud security, and data security. (*qduyujia@gmail.com*)

Guo-dong Wang, born in 1980, is an associate professor in the College of Information Engineering at Qing University, China. His research interests include network security, variational image Science, 3D reconstruction and medical image processing and analysis. (*doctorwgd@gmail.com*)

Zhen-kuan Pan, born in 1966, is a Professor and Ph.D. supervisor in the College of Information Engineering at Qingdao University, China. His main research interests include virtual reality technology, computer vision, image science, network and information security, embedded system and trust computing. (*zkpan@qdu.edu.cn*)

