# Hybrid Testbed Enabling Run-time Operations for Wireless Applications

Kumiko Maeda, Keisuke Nakata, Takaaki Umedu, Hirozumi Yamaguchi,
Keiichi Yasumoto† and Teruo Higashino

Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
{k-maeda,k-nakata,umedu,h-yamagu,higashino}@ist.osaka-u.ac.jp

†Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
yasumoto@is.naist.jp

## Abstract

*In this paper, we propose a hybrid Testbed enabling Run-time Operations for Wireless network Applications (TROWA). TROWA is a wireless network emulator and its simulation core is based on the network simulator MobiREAL. TROWA can emulate packet transmissions in multi-hop wireless networks in real-time, with realistic mobility models of urban pedestrians. Also it provides APIs that allow real stations to connect with their simulated networks with a little modification of applications and protocols. One of the significant features of TROWA is the control interfaces to manipulate the movement of simulated nodes during the execution of applications. This enables performance analysis and algorithm validation in an efficient way. Additionally, TROWA can improve the accuracy of real-time simulation by prioritizing the simulation events. By several experiments for a VoIP application, we show the usefulness of TROWA.*

## 1 Introduction

Performance evaluation of mobile wireless network applications and protocols is a complicated task. To improve the fidelity of evaluation with low-cost environment, network emulators which allow to test real applications running on real stations have been presented [10, 11, 15–18, 20, 23]. However another issue needs to be considered; it has been proved in many literatures that in mobile wireless networks that are highly dynamic, lack of realism in node mobility affects the fidelity of performance evaluation [5, 6, 8]. Therefore, we may prepare realistic mobility models in simulation of mobile wireless networks, but are still faced with difficulty in designing certain kinds of test scenarios. A typical example is a routing strategy test; we may want to disconnect established routes by removing a relay node in various situations for checking the reliability of the route repair process. This scenario can be implemented by specific node placement and movement, however it is difficult to do so in any situation – *e.g.* for any number of nodes, placement, movement and geography, it may be impossible.

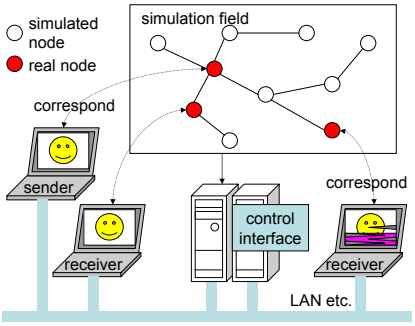



**Figure 1. Overview of TROWA architecture**

In this paper, we design and develop a new development environment called TROWA. TROWA enables the use of real application codes on real mobile terminals connected to a simulated network (see Fig. 1). Since the mobility models used in the simulated network include the Urban Pedestrian Flows (UPF) model [14], we may evaluate target applications under realistic city environments. TROWA contains a suite of "control interfaces", which manipulates the movement of mobile nodes, gives the user inputs to the target application in run-time, and visualizes the node mobility and network status during the simulation. Using these functions, for example, we can test to what extent video quality decreases in multicasting movies on MANET when

letting a multicast relay node move into a congested region. Also, we can verify how the file caching strategy works in a P2P file sharing application on MANET when we let a node walk around in the target area. We can visualize the files actually cached on the node to see how the strategy works in real environment. Through several experiments to run a VoIP application on two real terminals connected through a simulated MANET with the DSR protocol, we show the usefulness of TROWA.

TROWA offers APIs at the three layers (application, transport and network layers) to connect real terminals with the simulator. By using these APIs, protocol data units (PDU) issued by a real terminal of the corresponding layer are passed to the simulator, and the protocols under the corresponding layer are simulated. To realize real-time simulation, we prioritize packet processing events over the other events like node movement processing events. By this we can guarantee the timeliness of packets by sacrificing computation of node position updates and others.

Through the experiments, we show that TROWA can reduce the average delay in real-time simulation by up to 25% while keeping the location update error small enough. Also we have proved the usefulness of the facilities; node manipulation and real-time simulation with real applications.

## 2 Related Work

Network simulators such as ns-2 [2], OPNET [3] and Qualnet [4] have been widely used for the evaluation of network traffic. On the other hand, to test and evaluate the end systems themselves, many network testbeds have been developed [10, 11, 15–18, 20, 23]. StarBED [20] is a large-scale network testbed integrating simulation with emulation. They offer hundreds of physical nodes and can emulate thousands of virtual nodes. ORBIT [18] also offers about 400 inter-connected stationary wireless nodes that can emulate specified network topologies and traffics. Due to their features, they are mainly used to test the wide-scale (e.g. campus-scale, city-scale or Internet-scale) networks.

On the other hand, other types of testbeds like [10,11,17, 22] can be built in small spaces such as laboratory rooms and thus can easily be used to test end system applications. They mainly use physical nodes with some emulation techniques. In MobiEmu [22], using a packet filtering technique at the MAC layer, a testbed for implementing a given network topology is constructed. Mobile Emulab [11] takes a direct approach that uses mobile robots to move terminals physically. In MiNT project [10], a miniaturized wireless network testbed is constructed. In these testbeds, the accuracy of packet transmission is a benefit, however the scale and topology of target networks are limited by the number of physical nodes, their capability of movements, and space to deploy these nodes.

The hybrid approach in which simulation is integrated into emulation is reasonable to balance the timeliness of packet processing, adaptivity to the scale and topology, and hardware cost. ns-2 [2] implements an emulation function that allows real nodes to connect to simulated networks. TWINE [23] is a hybrid emulation testbed for wireless networks and applications, which can adaptively combine simulated, emulated and physical subnets. Based on this, a WHYNET framework is presented in [21] and several case studies are given. MobiNet [15] enables to test IP-based unmodified applications by emulating multi-hop wireless networks over a LAN cluster.

Our approach is close to the hybrid approaches, however it differs in the following points. We mainly focus on highly dynamic mobile wireless networks where node mobility strongly affects the network performance [5, 6, 8]. In such a network, application developers may want to test application performance with many mobility scenarios and geographical environments. To help those developers, we provide such a system that can simulate the realistic behavior of nodes based on the network simulator MobiREAL [1] and can control the movement of nodes. Also these operations can be done through GUIs. By this feature, developers can easily generate intentional route disconnection in testing routing protocols and many other situations that are hard to predict in advance. We have presented the results from case studies considering real-world applications.

In addition, we consider the priority of events so that events regarding packet transmissions can be prioritized to those regarding node mobility to realize real-time simulation. In such a situation, we have evaluated the trade-off between the timeliness of packet scheduling and node location accuracy. There are some existing researches that apply parallel distributed simulation to the network emulator [7,19]. The network simulator MobiREAL also supports parallel discrete event simulation [13]. As for a part of our future work, we are planning to apply our event prioritizing method to the parallel discrete event simulation to improve the timeliness of the scalable network simulation.

## 3 Overview of TROWA

We show the overview of the TROWA architecture in Fig. 1. In TROWA, real mobile terminals are connected via low delay networks (*e.g.* LAN) to a single host that simulates its network behavior. Hereafter, we distinguish two types of nodes in the simulated networks, *simulated nodes* and *real nodes*. Simulated nodes appear in the simulation field as shown in Fig. 1 where their entire components are simulated, while in real nodes their application codes and higher layer protocols are executed on real stations. These applications and protocols on real nodes use TROWA APIs to interact with simulated lower layers of the node instances

assigned to those real terminals so that they can communicate with simulated nodes and vice versa.

TROWA uses TCP communication with the socket library for cooperation between real terminals and the network simulator. It does not modify its OS dependent libraries nor limit the environment of real terminals. We offer three layers' APIs, the application layer API, the transport layer API and the network layer API. We specify one of the above APIs used by each real terminal. If a protocol data unit (PDU) of the specified layer is transmitted as data of the socket communication from the real terminal, the PDU is translated into the usable format in the simulator and is passed to the corresponding layer of the assigned node instance. In the case of using the application layer API, the real application sends the destination IP address, type of transport layer protocol, data to be transmitted and so on to the simulator, then the simulation of the transmission is initiated from the transport layer. If the node instance assigned to the real node receives a PDU of the specified layer, the PDU is transmitted to the corresponding real terminal.

## 3.1 Visualization and System Control

TROWA provides a set of control and visualization interfaces (called *control interface* for simplicity) which have the following three functions: (1) visualization of simulation status, (2) run-time manipulation of node movement, and (3) remote operation of applications on real terminals. We illustrate how multiple components interact to provide these functions in Fig. 2.

The control interface runs on the Microsoft Windows platform. The control interface and the simulator cooperate by exchanging control data over TCP connections. Therefore, the simulator and the control interface can run on separate computers. For example, we can execute the simulator on a high-performance workstation and the control interfaces on a consumer PC.

### 3.1.1 Visualization

In the main window of the visualization tool, the movement of nodes, wireless links, packet propagation and others can be visualized. We can choose whether these objects are to be displayed or not during simulation. It can also show some metrics like node density and packet loss rates measured in each region separated into grid cells.

### 3.1.2 Manual Operation of Mobility/Application

We can control movement of nodes manually through the control interface. We can change the positions, speeds and directions of nodes and add/delete nodes. In TROWA, the movement of a node is represented as a sequence of *waypoints*, each of which consists of (x,y)-coordinates, pause duration and velocity. During execution, waypoints can be added, deleted, or modified.

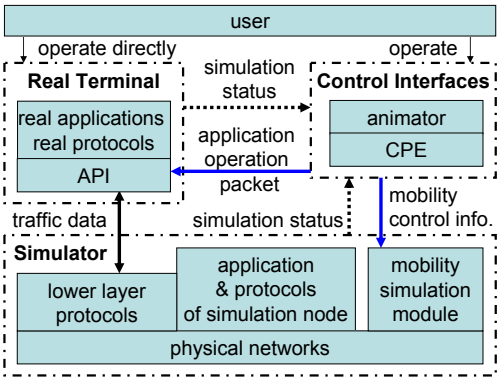



**Figure 2. Component Interaction in TROWA Control Interface.**

For usability, it is desirable to operate applications running on plural real terminals from a single interface. One way is to let the application programs and the control interface exchange *application operation packets* through the TROWA APIs, in parallel with data packet exchange. This packet includes commands to operate the applications. In this case we may need to implement a simple interpreter that understands the specified commands.

Another way is to prepare general GUIs like buttons and forms in the control interface. Although the applications are limited to Java programs, we have decided to use Javassist [9] to edit binary codes of Java programs. Javassist allows to change class definition, add fields and methods, change methods and so on without deep knowledge on the binary programs. Thus we can replace the common class GUIs with the class which can remotely be operated by the control interface through the TCP channel used by TROWA APIs. When we click a button on the control interface, the application operation packet is sent and the application program would behave as if the local button on the real terminal were clicked.

### 3.1.3 Automatic Operation of Mobility/Application

In TROWA, the human behavior of commands/data input to applications can be emulated. We use Condition Probability Event (CPE) model [12] as their user description model and a part of mobility. If one of the methods described in Section 3.1.2 is applied to the application program, we can automatically operate the application according to the CPE model. CPE model is a rule-based behavior description language and can represent such a situation that mobile nodes (or their users) dynamically change their behavior according to their environments. For details, see [12].

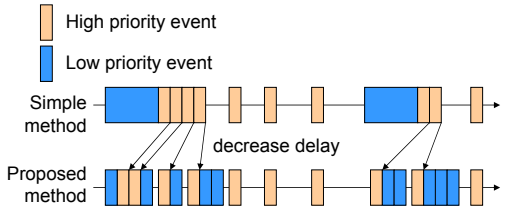



**Figure 3. Overview of event processing method.**

## 4 Real Time Simulation

In this section, first we explain how to realize real-time simulation. Then we show our experimental results on the accuracy of the synchronization.

### 4.1 Implementation

For real-time simulation, it is required to synchronize the simulation clock with the real clock, and each simulation event like packet transmission must be executed at accurate time. In parallel, TROWA simulates node mobility, however this mobility simulation may increase its processing load and may have bad influence on real-time simulation. Many realistic mobility models and some random-based mobility models need periodic updates of all nodes' positions. This position update does not require so much computation power. However, if this update is scheduled to the same time with packet processing events, they may cause temporary delay of event processing and we may lose timeliness.

In order to solve this problem, we propose a method to reduce the temporary delay of the simulation clock by prioritizing the execution of packet processing events. We classify simulation events into *high-priority events* that need strict timeliness like packet transmissions and *low-priority events* that are allowed to be delayed for a while like mobility calculation, mobility trace processing, statistical data processing and so on. Then we fragment low-priority events into smaller processing units and execute each unit when there is no high-priority event to be processed.

First, we explain how real-time simulation progresses in TROWA. TROWA adopts Discrete Event Simulation (DES). In DES, simulation progresses by executing events in the order of their timestamps. All events are stored in the event queue in the ascending order of the timestamps. Then the event at the head is dequeued and executed and new events generated by this execution are queued.

In TROWA, we use the following simple scheme to let the simulator clock synchronize with real time. First, the simulator gets the real clock value $T_0$ when the simulation starts (the simulator clock is 0 at this moment). Then the simulator executes the next event as soon as the current value $T_{now}$ of the real clock satisfies $T_{now} - T_0 \geq S_{next}$ where $S_{next}$ is the timestamp of the next event.

As seen above, in real-time simulation each event should occur along the real clock and this differs from DES which processes each event right after the previous event. Therefore, in the proposed method, we use the waiting time between two events to execute low-priority events. We show the overview of the proposed event processing method in Fig. 3. In the proposed method, the simulator fragments each low-priority event into smaller processing units. Each processing unit should be small enough to be able to expect that its processing time is not greater than a certain threshold $\delta$. Then the simulator executes each unit if residual waiting time is larger than $\delta$. Even in this scheme, in order to avoid too large delay of low-priority events, we should set their appropriate deadlines and prioritize them when their deadlines are approaching.

To handle the PDU transmitted from real terminals, sockets of real terminals are polled after execution of each simulation event, and process PDUs immediately if there are received PDUs. In additionCif there is no event that can be processed, the simulator waits until the processing time of the next event by watching the sockets.

We think that our proposed method prioritizing simulation events can be applied to the existing parallel and distributed simulation techniques. It is important to consider that parallel simulation makes some processes (machines) wait for synchronization with other processes. This means that if some processes delay their low-priority events, it may make synchronization interval longer. Solving this problem in TROWA is part of our future work.

### 4.2 Experimental Results

In order to evaluate the proposed simulation method, we conducted some experiments to validate the number of nodes that can be simulated in real time and the accuracy (timeliness) of the event processing time.

In the experiments, we regarded mobility calculation events as low-priority events. We measured the delay from the real clock for each high- or low-priority event with and without the proposed method. Since our method sacrifices the preciseness of node positions to a certain degree by delaying position update events, we also measured the error of the nodes' positions to see the influence.

We used an ordinary PC (Pentium4 3.40GHz, 2GB memory) for executing the simulator and two laptop PCs (PentiumM 1.6GHz, 1.5GB memory) as real terminals for executing application programs. In the experiment, the one real application send 1024 bytes data at 0.777s interval to the other application. The fixed simulated nodes are placed

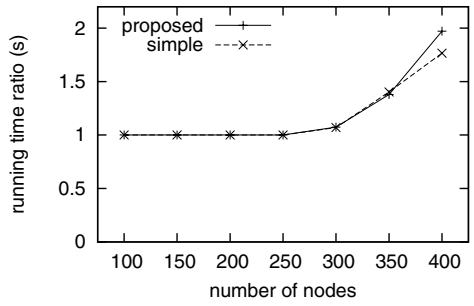


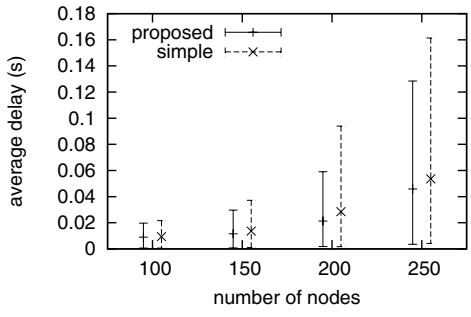

**Figure 4. Running time of 100s simulation.**

to let the simulated communication path between the real nodes be 3 hops. In addition, we also placed simulated nodes which follow the trace mobility with the 1 sec. interval granularity (moving nodes). Those nodes send single-hop broadcast packets at the interval randomly chosen between 10 sec. and 20 sec. The transmission range was set to 100 meters and the duration of the scenario was 100 sec. We used the application layer API. The size of the field was 500m $\times$ 500m. The deadline of the mobility calculation event was set to 1 sec., so each position update event was allowed to be delayed just before the next position update event.

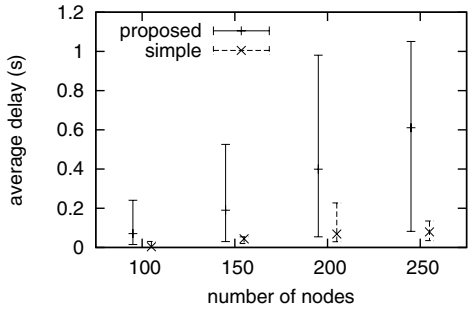
In Fig. 4, we show the ratio of the actual running time (i.e., time to complete simulation in real clock) to the specified simulation time changing the number of moving nodes from 100 to 400. In the figure, "proposed" and "simple" represent the proposed method which executes low-priority events during waiting time and "simple method" which executes all events in the order of their timestamps, respectively. Fig. 4 suggests that with this simulation scenario we can simulate up to 250 nodes in real time. We can see a little overhead of the proposed method for over 300 nodes.

Next, in Fig. 5, we show the average delay and confidence interval (95%) of the time when the events were actually processed from the real clock, changing the number of nodes. The figure shows that the proposed technique can reduce the average delay of the high-priority events by up to 25% by delaying low-priority events. For the 300 nodes case, the delay of the high-priority events of "proposed" and "simple" were 2.2 sec. and 1.7 sec., respectively. As a result, in our current implementation and given settings, networks with up to 250 nodes can be simulated in real-time without serious delay.

Fig. 6 shows the distribution of packet arrival interval to the application at the destination (real terminal). 'simulation' is the distribution of packet arrival interval in the simulation time when we use the sequential (not real-time) simulation and the application implemented in the simulator instead of real application. The number of moving nodes




(a) Delay of high-priority events




(b) Delay of low-priority events

**Figure 5. Delay of event processing clock from real clock.**

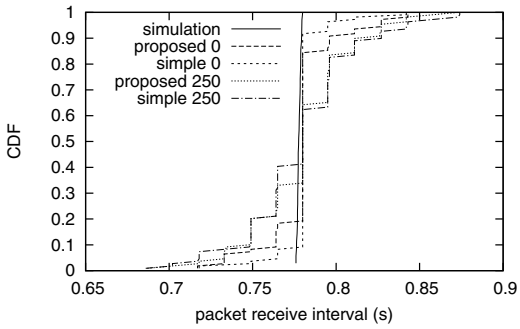



**Figure 6. Distribution of packet arrival interval.**

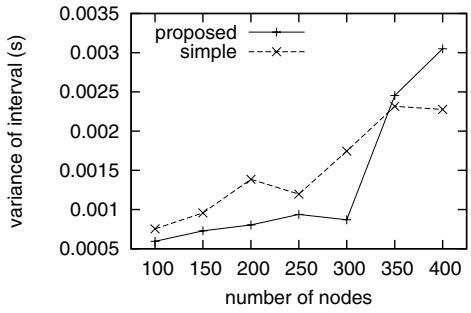



**Figure 7. Variance of packet arrival interval.**

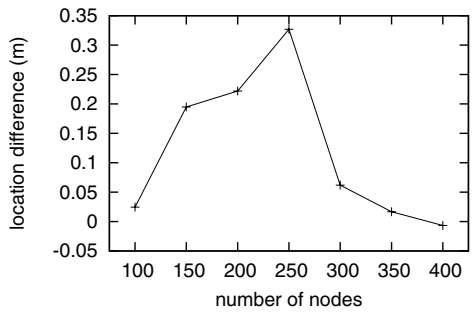



**Figure 8. Average distance of the node location.**

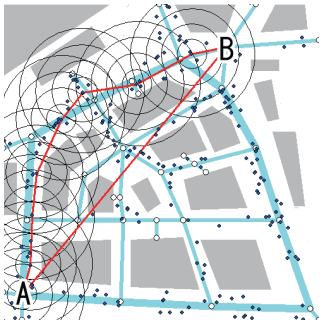



**Figure 9. Snapshot of visualization tool.**

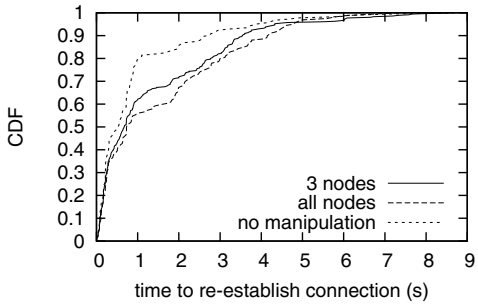



**Figure 10. Cumulative Distribution of Route Repair Time**

was set to 0 or 250. We assume that 'simulation' is the ideal distribution. The difference of 'proposed' and 'simple' from 'simulation' depends on the delay of high-priority events in Fig. 5. Also, Fig. 7 shows the variance of packet arrival interval of the above case. The ideal value ('simulation' in Fig. 6) is close to 0. We can see that our proposed method reduces the variance.

Next, we show the average distance of the moving nodes' positions between with and without the proposed prioritizing method in Fig. 8. The error was sampled just before the update of the node positions in the proposed method. We can say that the difference is really small compared with the transmission range (100m), so the impact of this difference on the accuracy of simulation results is expected to be quite small.

## 5 Case Study

In order to show the usefulness of two features of TROWA, which allow us (1) to change the behavior of specified nodes in run-time and (2) to simulate wireless network with real stations, we have conducted some experiments to run a VoIP application over the DSR protocol on MANET.

### 5.1 Experimental Configurations

We used a map of actual urban area around Osaka Station in Japan as the simulation field as shown in Fig. 9. The size of the field was 500m $\times$ 500m. The total number of nodes was about 200 and the speed of each node was 1.1 – 1.7 m/sec. Moreover, we reproduced realistic pedestrian flows in urban areas based on the actually observed density on each street using the Urban Pedestrian Flow mobility model [14]. During the simulation, VoIP communication data was transmitted at 9.6kbps. The radio range was set to 100 meters. We used IEEE802.11 DCF with the RTS/CTS mechanism as the MAC protocol.

### 5.2 Intentional Route Break in DSR

In this experiment, we create the situation that was difficult to create without run-time manipulation in order to show its usefulness. We evaluated the recovery time from the DSR route break with the UPF mobility.

In the experiment, CBR traffic is generated from the node at point A in Fig. 9 to the node at point B. We stop the simulation temporarily using the GUI of TROWA when the route is stable, delete 3 or all intermediate nodes, and restart the simulation. By this we can simulate the situation that multiple nodes disappear at the same time. If we use somewhat complex mobility model to improve the realism, it is difficult to know in advance on which nodes the route is established due to dynamics of network topology and unpredicted movement of nodes in real environment. To search the intermediate node in run-time, we may have to add some function to the simulator, but we can soon find the intermediate node and can delete the node with GUI of TROWA.

Fig. 10 shows the cumulative distribution of route repair time. In the graph, "3 nodes" represents the case where three relay nodes on the DSR route were chosen randomly and deleted, and "all nodes" represents the case where all the relay nodes on the DSR route were deleted. Also for

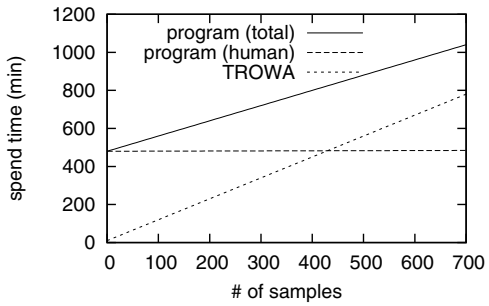



**Figure 11. Time to collect data of Fig. 10.**

comparison purpose, we have evaluated an additional case; "no manipulation" represents the case where no manipulation was applied during simulation. We can see clearly that disappearance of all the intermediate nodes makes the route repair time longer. Also, we can see that the repair time of "no manipulation" is smaller than the other two manipulation cases. From this result, we can expect that in the original node mobility without our operation, such a situation that forwarders suddenly and simultaneously disappear cannot be observed. The run-time operation function of TROWA enables us to give some intentional factor into the on-going network simulation according to the visualized current status. Next, we show an example to explain how much TROWA helps simulator users in such a case.

To use network simulators, we need certain amount of experience and much effort. To see how TROWA helps to alleviate this burden, we have asked three master course students in our laboratory to implement the function that automates the manipulation in the previous experiment into the network simulator. The function finds the intermediate nodes and deletes those nodes at the same time. The student "A" having an experience of modifying the simulator took about 4 hours, the student "B" took about 12 hours, and the student "C" gave up after 4 hours. It seems to be hard for inexperienced users to understand the source files of the DSR protocol that have about 1600 lines including the header files. Of course, this result is only an example, but one can see that the implementation is not an easy task as everyone can finish in a short time.

Fig. 11 shows an example of the time to collect data used to obtain the results in Fig. 10 against the number of the collected data samples. We show the time of two data collection methods. One method is to implement the manipulation function as explained above into simulator codes. In this case, we need time to implement the function, which we set to 480 minutes (this is the average time of two students "A" and "B") in addition to the time to run the simulation. "program (human)" in the figure represents the time which we have to spend for the experiment and most of the time is for the implementation. The case "program (total)" represents the total time to collect samples and this includes the running time of the simulation when we can spend our time for other things. Another method is to use the TROWA function and is "TROWA" in Fig. 11. In this case, we do not need to write codes. Instead we directly specify which nodes should disappear through GUI so we have to spend our time while running the simulation.

Even though TROWA offers GUIs, we need to operate the GUIs at any moment when we need to change situation. Therefore, it might take much effort if we need to collect large amount of data. For example, as shown in Fig. 11, the time to collect samples by TROWA exceeds the time required to prepare the program if we collect 420 or more samples. However the advantage of TROWA is that we can easily simulate application/protocols under various situations. For example, before we conduct the detailed performance evaluation, we may want to test the application with a wide variety of configurations and find the best configuration in which we can examine the application's various aspects. In general preparing all of the configurations may take long time and require a lot of efforts, thus we provide an environment where we can set and modify configurations through UI observing visualized simulation status. Consequently we may reduce the cost of the trial and error process required in conventional simulations. This advantage can improve the efficiency of the design and the development process.

### 5.3 Voice Quality over VoIP

In order to show the effectiveness of using real application programs in terms of utilizing users' perception for the design of the application, we have asked six students to make conversations through the real terminals in the context of the VoIP application.

It is a common technique for VoIP client software to buffer the voice data to salvage the delayed packets. When we set the longer time for buffering, the data loss ratio becomes smaller and thus the playback quality of the voice will be better, whereas the end-to-end delay becomes larger and the perceptive delay quality will be worse. There is a trade-off between playback quality and delay depending on buffer size. In the experiment, changing buffer size in the application program, we measured the loss ratio of voice data and obtained subjective feedbacks from the students.

Fig. 12 depicts frequency that voice data was not received for the interval longer than the specified time duration (0.25 sec to 4 sec in x-axis) on the 8 hop route varying the buffer size of the VoIP application. The figure shows that the buffering can clearly reduce the frequency of longer intervals with no voice. Thus, the playback quality should be improved.

To confirm the degree of this improvement, we con-

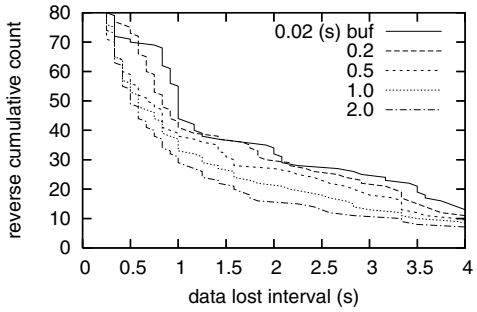



**Figure 12. Distribution of data loss interval.**

**Table 1. Perceptive voice quality depending on buffer size**

| buffered time (s) | 0.02 | 0.2 | 0.5 | 1.0 | 2.0 |
|---|---|---|---|---|---|
| Student A | 1 | 1 | 2 | 4 | 4 |
| Student B | 1 | 1 | 1 | 4 | 4 |
| Student C | 1 | 1 | 1 | 3 | 3 |
| Student D | 1 | 1 | 2 | 3 | 4 |
| Student E | 1 | 1 | 4 | 4 | 3 |
| Student F | 2 | 5 | 5 | 4 | 3 |

1: terrible, 2:bad, 3:tolerable, 4:acceptable, 5:good

ducted a questionnaire to six students where each student played back voice transmitted through 8-hop connection, under five settings of the buffering time. The results are shown in Table 1. Most students recognized the improvement of the voice quality when the buffering time was 1.0 or larger. Two students preferred 1.0 sec buffering time than 2.0 sec due to the trade-off between the delay and the playback quality.

Although we can measure traffic characteristics like data loss ratio and end-to-end delay by network simulators, it is difficult to grasp human perception from the measured values. In the above experiment, the appropriate value for the buffering time strongly depends on the user's perception. Therefore, it is quite important to evaluate such real-world performance and develop applications considering it.

## 6 Conclusion

In this paper, we have proposed a new real-time wireless network emulation environment called TROWA for the design and performance evaluation of mobile wireless network systems. In TROWA, application programs on real mobile terminals can be tested and verified through the simulated network. In addition to this real-time simulation facility, the one of the important feature of TROWA is that it allows to modify the movement of nodes and to operate the real application programs during the simulation, observing the locations of nodes and network conditions. Thus we can readily create various situations and can try them by removing implementation process that may have a large cost in some cases. Actually, we have evaluated this situation in our experiment, as well as another experiment which tested user perception on VoIP quality over MANET.

As future work, we are planning to increase scalability of TROWA emulator engine. The MobiREAL simulator, part of its functions are incorporated into TROWA, supports parallelized DES [13]. By combining the DES and proposed event prioritizing method, we try to speed up event processing. We are also planning to open TROWA to public domain so that many developers can enjoy their benefit from our design concept and software.

## References

[1] *MobiREAL Simulator Web Page*. `http://www.mobireal.net/`.

[2] *ns-2*. `http://www.isi.edu/nsnam/`.

[3] *OPNET*. `http://www.opnet.com/`.

[4] *QualNet*. `http://www.scalable-networks.com/`.

[5] F. Bai, N. Sadagopan, and A. Helmy. The IMPORTANT framework for analyzing the impact of mobility on performance of routing for ad hoc networks. *AdHoc Networks Journal*, 1(4):383–403, Nov. 2003.

[6] J.-Y. L. Boudec and M. Vojnovic. Perfect simulation and stationarity of a class of mobility models. In *Proc. IEEE Infocom*, volume 4, pages 2743–2754, 2005.

[7] R. Bradford, R. Simmonds, and B. Unger. A parallel discrete event ip network emulator. In *Proc. 8th IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '00)*, pages 315–322, 2000.

[8] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[9] S. Chiba and M. Nishizawa. An easy-to-use toolkit for efficient java bytecode translators. In *Proc. 2nd Int. Conf. on Generative Programming and Component Engineering (GPCE '03)*, pages 364–376, 2003.

[10] P. De, A. Raniwala, S. Sharma, and T. cker Chiueh. MiNT: A miniaturized network testbed for mobile wireless research. In *Proc. IEEE Infocom*, volume 4, pages 2731–2742, 2005.

[11] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau. Mobile emulab: A robotic wireless and sensor network testbed. In *Proc. IEEE Infocom*, 2006.

[12] 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. 13th IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 499–502, 2005.

[13] K. Konishi, H. Yamaguchi, and T. Higashino. Efficient parallel simulation of mobile wireless networks by run-time prediction of multi-hop propagation delay. In *Proc. 3rd IEEE Int. Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2007)*, 2007.

[14] 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. 8th ACM/IEEE Int. Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 151–158, 2005.

[15] P. Mahadevan, A. Rodriguez, D. Becker, and A. Vahdat. MobiNet: a scalable emulation infrastructure for ad hoc and wireless networks. *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)*, 10(2):26–37, 2006.

[16] D. Mahrenholz and S. Ivanov. Real-time network emulation with ns. In *Proc. of DS-RT'04*, 2004.

[17] E. Nordstrom, P. Gunningberg, and H. Lundgren. A testbed and methodology for experimental evaluation of wireless mobile ad hoc networks. In *Proc. 1st Int. Conf. on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM 2005)*, pages 100–109, 2005.

[18] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In *Proc. of IEEE WCNC 2005*, 2005.

[19] R. Simmonds, R. Bradford, and B. Unger. Applying parallel discrete event simulation to network emulation. In *Proc. 14th workshop on Parallel and distributed simulation (PADS '00)*, pages 15–22, 2000.

[20] K.-i. C. Toshiyuki Miyachi and Y. Shinoda. StarBED and SpringOS: Large-scale general purpose network testbed and supporting software. In *Proc. of Valuetools 2006*, 2006.

[21] M. Varshney, Z. Xu, S. Mohan, Y. Yang, D. Xu, and R. Bagrodia. WHYNET: a framework for in-situ evaluation of heterogeneous mobile wireless systems. In *Proc. of ACM WinTECH '07*, pages 35–42, 2007.

[22] Y. Zhang and W. Li. An integrated environment for testing mobile ad-hoc networks. In *Proc. ACM MobiHoc*, pages 104–111, 2002.

[23] J. Zhou, Z. Ji, and R. Bagrodia. TWINE: A hybrid emulation testbed for wireless networks and applications. In *Proc. IEEE Infocom*, 2006.