DynamiQ: A Tool for Dynamic Emulation of Networks

Razvan Beuran*
Japan Advanced Institute of Science and Technology
Japan
razvan@jaist.ac.jp

Shingo Yasuda
National Institute of Information and Communications Technology
Japan
s-yasuda@nict.go.jp

Tomoya Inoue
Japan Advanced Institute of Science and Technology
Japan
t-inoue@jaist.ac.jp

Yuuki Takano
National Institute of Information and Communications Technology
Japan
ytakano@nict.go.jp

Toshiyuki Miyachi
National Institute of Information and Communications Technology
Japan
miyachi@nict.go.jp

Yoichi Shinoda
Japan Advanced Institute of Science and Technology
Japan
shinoda@jaist.ac.jp

ABSTRACT

Interactive network experiments, in which experiment conditions change dynamically based on input from users or other external sources, are the most appropriate approach when evaluating solutions to practical network problems, for teaching and/or training purposes, etc. Support for dynamic experiment conditions is also required whenever an experiment cannot be fully defined from start, for instance when node behavior (application execution, mobility pattern, etc.) depends on factors such as communication conditions in the experiment, traffic content, and so on.

In this paper we present the network emulation module named `dynamiQ` that makes possible the dynamic emulation of networks. We also outline an interactive experiment framework that uses `dynamiQ` to meet the above requirements. The evaluation of `dynamiQ` in this context shows that no significant performance penalties occur because of its dynamic nature. Our interactive experiment framework has already been used in practice, including for a demonstration at Interop Tokyo 2014.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Experimentation, Performance, Verification

*Razvan Beuran is also with the National Institute of Information and Communications Technology (NICT), Japan. This work was done when he was mainly with NICT.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
TRIDENTCOM '15 Vancouver, Canada
Copyright 2015 ACM X-XXXXX-XX-X/XX/XX ...$15.00.

Keywords

Dynamic experiment conditions, network emulation, interactive experiments

1. INTRODUCTION

Traditionally, network experiments are carried out based on predefined scenarios, especially in the case of simulation and emulation. This is the most straightforward approach, but it requires knowing/deciding in advance all the experiment conditions: number and position of the nodes (including their mobility if any), properties of the communication environment, traffic pattern and content, etc.

Predefined scenarios are however not feasible in circumstances such as the following ones:

- User input is required or desired for performing an experiment;
- Experiment conditions depend on external inputs, such as physics simulators, etc.;
- The behavior of experiment components depends on the node communication pattern and/or content.

Interactive experiments are the most appropriate alternative when searching for the solution to a certain practical problem, such as the wireless network planning for a certain geographic area. Thus, network experts could use an interactive experiment platform to virtually deploy various network solutions and analyze their cost and performance characteristics, without the time overhead, risks and expenses related to the *actual deployment* of alternative solutions.

Interactive experiments can also be used for teaching or training purposes, as a means to provide an affordable yet valuable hands-on experience. For example, students can be asked to construct networks in the virtual environment, and then investigate the state of the routing protocols involved. Given the use of emulation, this can be done in the same way as in the real world, by logging into the running machines and analyzing their state.

Complex experiments typically require data from external simulators in order to recreate realistic conditions. For instance, a physics simulator could be used to determine the

effects of an earthquake or some other disaster on a certain geographic area. These effects, such as damage to the network and road infrastructure, need to be fed into the experiment scenario, so that their influence is taken into account (e.g., nodes become offline, mobility patterns change, etc.). The easiest way to handle such situations is to support dynamic experiment conditions in the experiment framework.

Some network experiments may necessitate the ability to reproduce changes in nodes' behavior depending on their communication activities. For instance, based on the information received, nodes may decide to change their mobility pattern (e.g., modify their trajectory in order to avoid congested/unusable roads – a typical application of vehicular networks). The easiest manner to handle such kind of scenarios is again to support dynamic experiment conditions.

In this paper we present our endeavor towards creating an interactive experiment framework that makes it possible to perform realistic network experiments in dynamic, externally-controlled conditions. The main contributions of this paper are:

- Describe the dynamic network emulator that we designed and implemented, named `dynamiQ`, which is a key element in this context;
- Discuss the overall interactive experiment framework that uses `dynamiQ` to provide the above functionality.

Our interactive experiment framework has already been used in practice, including for a demonstration entitled "Interactive Mesh Network Planning" at the Interop Tokyo 2014 annual trade fair for information technology, where it received the Special Jury Best of Show Award.

The remainder of the paper is organized as follows. In Section 2 we summarize the most important aspects regarding the network emulation testbed used in our approach. Section 3 introduces the dynamic network emulation tool `dynamiQ` that was developed based on several existing testbed components. In Section 4 we outline the design of the interactive experiment framework that uses `dynamiQ`. This is followed in Section 5 by a performance evaluation of `dynamiQ` in the context of the interactive framework implementation. The paper ends with conclusions and references.

2. NETWORK EMULATION TESTBED

An interactive experiment framework such as the one we propose requires an execution platform with sufficient capabilities to meet all the execution requirements. While simulators can be used for some aspects of the framework, for the network-related aspects we use the emulation approach, since it allows using real applications and protocols in the experiments. Thus, the results and observations are directly applicable to practical situations.

Another important characteristic of emulation is that it is by design intended to run in real time, a necessary feature when dealing interactively with user input.

2.1 Infrastructure

StarBED is a network testbed located in Ishikawa Prefecture, Japan, at the Hokuriku StarBED Technology Center of the National Institute of Information and Communications Technology (NICT) [6].

StarBED makes available for experiments more than 1400 interconnected PCs, and represents the physical infrastructure of our interactive experiment framework. The control and the experiment networks of StarBED are independent of each other, so as to prevent interference between the management and the experiment traffic.

SpringOS is a software suite employed for performing experiments on StarBED. SpringOS is used to configure the hosts and the interconnecting network for the experiment, and also to effectively run the experiments.

2.2 Network emulation

QOMET (Quality Observation and Mobility Experiment Tools) is a set of tools for network emulation targeting mainly wireless networks [1]. The input of QOMET is represented by an XML-based user-defined scenario that describes the network environment, including node properties and mobility, area topography, etc.

QOMET provides the necessary mechanisms for performing experiments in a distributed manner by reproducing on StarBED the communication conditions between the emulated wireless nodes [2]. The network emulation component of our interactive experiment framework, `dynamiQ`, is based on QOMET, as it will be described in Section 3.

2.2.1 deltaQ

As discussed in detail in the references mentioned above, emulation using QOMET is performed as a two-stage process. Firstly, the module called `deltaQ` will process the input scenario and compute the communication conditions between nodes, which we call *network quality degradation* and denote by ΔQ.

The ΔQ parameters are calculated as follows (we use here IEEE 802.11a/b/g as an example, but other network technologies are handled in a similar manner):

Packet loss rate: The frame error rate (FER) at PHY layer is computed based on the distance between the communicating nodes, the properties of the communication environment, propagation model, etc. Then the MAC layer retransmission mechanism is taken into account to determine the network layer packet loss rate;

Packet delay: It is calculated as an average value by considering physical aspects (transmission and propagation delays), but also the actual MAC layer protocol and its specific mechanisms, such as congestion avoidance and retransmission;

Available bandwidth: Knowing the packet duration, current operating rate and packet delay, it is straightforward to determine the available bandwidth.

The computed ΔQ parameters are saved in a binary format that will be used subsequently during the real-time emulation process.

2.2.2 wireconf

In the second stage, the QOMET module called `wireconf` will recreate in the StarBED experiment network the communication conditions between scenario nodes based on the ΔQ parameters previously calculated. This is achieved through the use of the link emulation system named `ipfw`, which can artificially introduce packet loss, delay and bandwidth limitations for the configured traffic flows [3].

First of all `wireconf` does some preliminary initialization of `ipfw` depending on the relevant parameters read from the

deltaQ output file, mainly related to the creation of the appropriate rules and pipes for controlling the communication conditions between nodes. Following that, the operation of wireconf involves the following steps executed periodically based on a user-defined time interval:

1. Read the ΔQ parameters that correspond to the next time interval from the deltaQ output file;

2. When the deadline for changing network conditions arrives, reconfigure the ipfw pipes according to the ΔQ parameters.

The above loop is executed until all the parameters in the deltaQ output file are exhausted, and the emulation ends.

3. DYNAMIC EMULATION SUPPORT

In order to support dynamic emulation of networks it is necessary to have a network emulation component that can emulate the experiment network even as scenario conditions change. Such dynamic scenario reconfiguration ensures that interactive experiments under direct user control can be performed (although some predefined network components can be included to facilitate experiment setup).

Scenario reconfiguration must be handled in real time, so that both the internal scenario changes and the communication condition changes are applied in a timely fashion. Unless this happens, the user cannot get the necessary feedback from the experiment framework, which diminishes considerably the advantages of interactive execution.

3.1 Overview

The solution that we have chosen is that each node in an experiment handles its own network emulation functionality. This makes it possible to increase experiment scale with a linear $O(N)$ dependency on the number of nodes (compared to the case when emulation management would be done in a centralized manner, hence with an $O(N^2)$ complexity).

The dynamic network emulation functionality is implemented by the module called dynamiQ (dynamic QOMET). DynamiQ effectively provides the network emulation functionality of QOMET but in a dynamic manner, without a predefined scenario as its static counterpart, the original QOMET.

For this purpose an external component needs to be introduced, that we call *Experiment Manager*. Its function is to continuously provide information to dynamiQ regarding experiment scenario changes. This module and the other components of the overall interactive experiment framework will be discussed in Section 4.

Each node in the emulation experiment runs an instance of the dynamiQ module (see Figure 1). This module has to handle the following functions:

1. Communicate with the *Experiment Manager* for scenario (re)configuration purposes;

2. Compute the network communication conditions that correspond to the updated scenario;

3. Configure the network connection so as to recreate the newly calculated conditions.

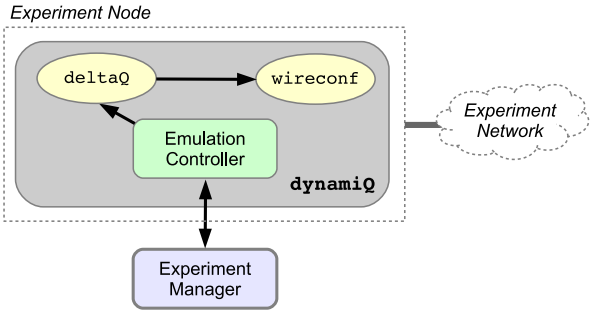

Figure 1: Architecture of the dynamiQ network emulator; an instance of dynamiQ runs on each experiment node.

3.2 Implementation

The functionality of dynamiQ is implemented as follows. The module maintains a data structure representing the experiment scenario: nodes, communication environment, etc. The core component named *Emulation Controller* contains a loop which is executed periodically based on a user-defined time interval. At each step of the algorithm the following actions are taken:

1. Request scenario information from the global *Experiment Manager*; the content is parsed, and the internal scenario representation is updated accordingly. For instance, new positions are set for the experiment nodes, new experiment nodes of the specified types are activated, node settings, such as transmit power, are changed, etc.;

2. After updating the scenario representation, call the deltaQ library of QOMET to recompute the communication conditions between nodes;

3. When the deadline for changing network conditions arrives, apply the new communication conditions to the network link by means of the wireconf library.

We decided to use the "pull" communication model in dynamiQ for retrieving scenario information from the *Experiment Manager* so as to have a controllable rate at which reconfiguration events occur. Thus, all the changes that take place during a time interval are dealt with at once, and emulation accuracy is not influenced by the computation burden incurred if changes were dealt with individually.

The dynamiQ module is implemented in C, similar to the other QOMET components. The module uses the JSON (JavaScript Object Notation) data format [4] to communicate with the *Experiment Manager*. For handling JSON data we used jsmn, which is a minimalistic JSON parser implemented in C [8].

4. INTERACTIVE EXPERIMENT FRAMEWORK

The interactive experiment framework that we designed has several components in addition to dynamiQ, as it will be described next. Please refer to Figure 2 for an overview of the architecture showing the components and their interactions. Note that the practical details in the discussion that

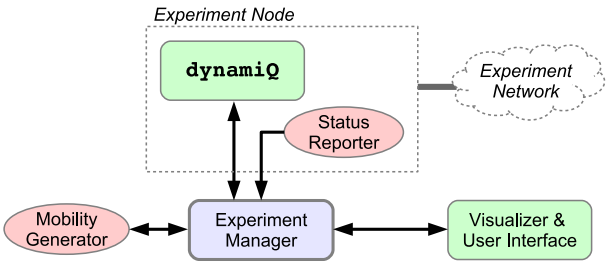

Figure 2: Overview of the interactive experiment framework architecture.

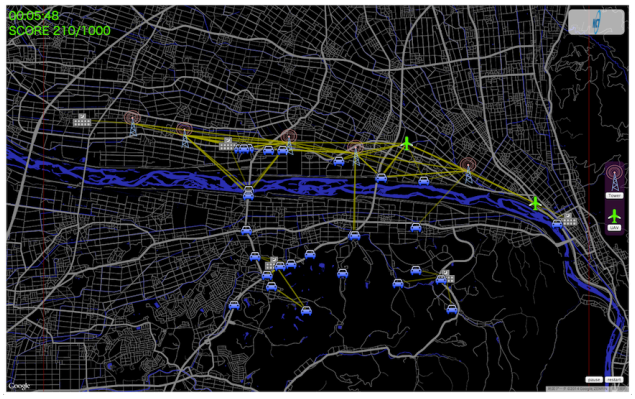

Figure 3: Screenshot of the 2D experiment visualizer and user interface.

follows are based on the current proof-of-concept implementation of the framework; some aspects may change in the final implementation.

4.1 Experiment Manager

The experiments in our framework are managed by a central module named *Experiment Manager*. Its role is to multiplex the information from different sources, and to drive the execution of the framework components.

The *Experiment Manager* is implemented in Ruby and uses the JSON format for exchanging information with all the other components.

4.2 Visualizer & User Interface

The *Visualizer & User Interface* module, as its name indicates, first of all handles the display of the experiment state to the user. Secondly, this component is used to get user input so as to control the experiment flow. Thus, one can add nodes to the experiment, modify their position or settings, and so on, then see how these changes affect the experiment, for instance, how the network topology and communication conditions vary.

In order to effectively deal with user input, experiment status must be visualized continuously. In our current implementation, a 2D visualizer is used to provide a top view of the experiment area (see Figure 3). The 2D visualizer is coupled with the user interface to provide input to the interactive framework. The current implementation uses JavaScript, so that it can run in any web browser. During demonstrations user input is typically provided through a touch-screen interface running Windows 8.

4.3 Other components

The modules described so far are fundamental components of the interactive experiment framework. However, for demonstration purposes, two more modules were added to our proof-of-concept implementation.

4.3.1 Status Reporter

The first of them is called *Status Reporter*. Its role is to gather status information from the running experiment nodes, hence one instance runs on each node. The status information is then used by the other framework components, at this time mainly for visualization purposes.

Status Reporter functionality focuses presently on obtaining the topology of the mesh network that we constructed in our demonstration experiments using the OLSR routing daemon named `olsrd` [7]. This is achieved by employing the `txtinfo` plugin of OLSR, which accepts HTTP connections and returns internal OLSR status information.

The information thus collected from the experiment network is converted to JSON format and sent to the *Experiment Manager*, which then makes it available to the *Visualizer & User Interface* module. As an optimization, in the current implementation the *Emulation Controller* component of `dynamiQ` also provides the functionality of the *Status Reporter*. This allows minimizing the communication overhead, since OLSR information is included in the "pull" requests sent by `dynamiQ` to the *Experiment Manager*.

4.3.2 Mobility Generator

The `deltaQ` module in QOMET includes some mobility generation capabilities, however our dynamic experiment framework makes it possible to interface with external simulators as well, and thus to easily extend the functionality of the framework.

In our proof-of-concept implementation we have used the ONE simulator [5] as a mobility generator, in particular we used the built-in SPMBM (Shortest Path Map Based Movement) model in the ONE simulator to generate trajectories of the mobile nodes given the road map on which they are to move and their destinations.

The *Mobility Generator* module drives the ONE simulator and provides the resulting mobility trace data to the *Experiment Manager* on demand. This module, in its turn, supplies mobility information to `dynamiQ` for network emulation purposes, and to the *Visualizer & User Interface* for display purposes.

5. PERFORMANCE EVALUATION

The interactive experiment framework that we designed and implemented has several advantages compared to traditional predefined experiment platforms, since it makes possible a both affordable and realistic hands-on experience with network experiments. However, some trade-offs are necessary to provide this functionality.

5.1 Time lag

The main potential issue with our dynamic emulation approach is that interactivity may introduce a time lag in the experiment flow. Factors the may contribute to this are:

- The time gap between the moment a user interacts with the framework through the user interface, and the moment this information is made available to `dynamiQ`;

- The additional delay until the corresponding actions are actually recreated in the emulated network.

The first time gap depends mainly on the communication delays between the *Visualizer & User Interface* and *Experiment Manager*, and between the *Experiment Manager* and `dynamiQ`. If the hosts running these systems are all in the same local network, we estimate this overhead should not exceed several tens of milliseconds.

For demonstrations, even if the experiment hosts are within the StarBED testbed, the user interface needs to be deployed at the actual location of the demonstration. In this case the communication delay and jitter between the *Visualizer & User Interface* and the *Experiment Manager* increase, and could have a significant effect on overall system responsiveness. To handle this issue there are two alternatives: (i) Use QoS solutions for minimizing the above delay and jitter; (ii) Use a "portable" mini-testbed instead of StarBED that is placed in the same network with the user interface host.

The additional delay related to reproducing the emulated network conditions is dominated by the time needed to recompute the communication conditions based on the reconfigured scenario. As mentioned before, given the distributed execution mechanism of `dynamiQ`, this delay has an $O(N)$ dependency on the number of nodes.

Our approach for dynamic emulation of networks works as expected if the sum of the above two delays introduced by the dynamic emulation framework does not exceed the time step used in emulation. Note that given the "pull" communication model used by `dynamiQ` to retrieve scenario information, a delay of up to one time interval can occur before user interface actions are taken into account. Since the time interval was set to 500 ms, this should not impact significantly the framework interactivity and responsiveness.

As a side note, users of our framework have noticed that it takes a certain time between performing an action (such as adding a new node to the network) and seeing its effect on the network topology (e.g., until the node becomes part of the network topology). This issue is unrelated to the discussion above, and it is a property of the underlying network protocols. For instance, it simply takes some time for a protocol such as OLSR to discover a new network node, to recompute the network topology, and to update its internal state. Such delays are normal, and are a consequence of the fact that we employ real network protocol implementations; an instantaneous effect may seem more "appealing" at first sight, but it is neither realistic nor meaningful.

5.2 Experimental evaluation

We present here an evaluation of the interactive experiment framework that focuses on the `dynamiQ` module and its interaction with the *Experiment Manager*.

The evaluation was done in the following conditions. The experiment infrastructure were hosts with Intel Xeon CPUs at 2.5 GHz and with 32 GB RAM running the Ubuntu 14.04.1 LTS operating system. The emulated nodes were executed on these hosts as virtual machines (VMs) running the CentOS 6.5 operating system. A total of 4 hosts were used for executing the VMs, each running up to 15 VMs, hence the experiment involved up to 60 emulated nodes.

The software installed on each VM contained all the modules necessary for network emulation, including `dynamiQ` and the *Status Reporter* module. The OLSR routing protocol used in the emulated network was installed as well.

Table 1: Statistics for the time lag related to `dynamiQ`.

Time lag [ms]	Min.	Avg.	Max.	Std. dev.
T_{Comm}	0.33	2.61	1003.32	13.62
T_{Comp}	0.01	0.26	12.82	0.16
T_{Total}	0.43	2.87	1003.94	13.62

The global framework components, namely the *Experiment Manager* and *Mobility Generator* were run on another host. This management host had the same characteristics and was located in the same local network with the experiment hosts in order to minimize the communication delays. The *Visualizer & User Interface* module was run on another host in the same local network, again so as to minimize the communication delays.

For experiment purposes we instrumented the `dynamiQ` module to measure the two critical time gaps discussed in Section 5.1: (i) the communication time between the *Experiment Manager* and `dynamiQ`, that we denote by T_{Comm}, and (ii) the computation time for communication conditions, denoted by T_{Comp}. We didn't consider the communication delay between the *Visualizer & User Interface* and the *Experiment Manager* since in general it depends on the location where the experiment takes place; moreover it is not related to the module on which we currently focus, `dynamiQ`.

The measurement results, obtained for a typical emulation experiment with a duration of 10 minutes, are shown in Table 1. It can be noticed that the average communication time is of the order of milliseconds, and its standard deviation is of about 13 ms. However, large values do occur occasionally, up to about 1 s, since communication between modules is done using the HTTP protocol over shared networks. On the other hand, the computation time in `dynamiQ` is more stable, as both the average and standard deviation values do not exceed 1 ms; occasional larger values are observed, but they are at most of the order of tens of milliseconds.

We have also calculated the total time lag, T_{Total}, as the sum of the above two delays. As expected based on the discussion above, the total lag is mainly influenced by the communication time. To demonstrate that large delays occur very rarely, we plot in Figure 4 the $CCDF$ (Complementary Cumulative Distribution Function) of T_{Total}, which is equal to $1-CDF$, where CDF represents the empirical Cumulative Distribution Function of the total time lag. Thus we determined that the probability to have a total time gap exceeding 20 ms is only 0.1%. We consider this acceptable given the 500 ms time interval between emulation steps.

5.3 Discussion

Scalability can be an issue with systems such as ours. From the network emulation point of view, we mitigate this through the distributed execution of `dynamiQ`, as mentioned already in Section 5.1. Based on the measurements in Section 5.2, we expect the network communication condition computation to be in the order of a couple of tens of ms at most even when the system scales by a factor of 10.

From the interactivity point of view, the only solution is to minimize the network delays between the components of the system. Our performance evaluation in Section 5.2 has shown that when all the components are in the same local network there is no significant issue in this respect. Given that we only used 5 physical hosts for those measurements,

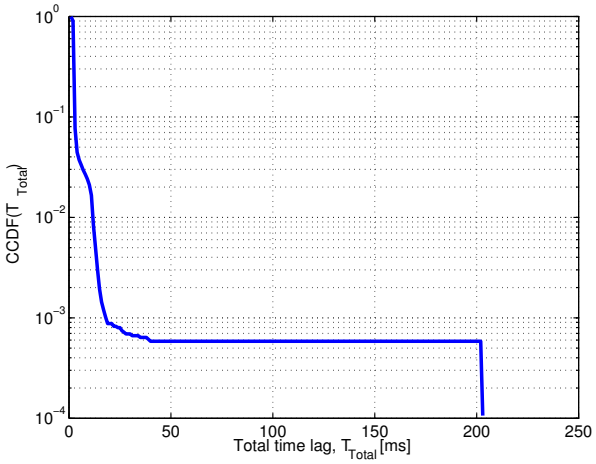

Figure 4: Total time lag for `dynamiQ` execution (plotted using the $CCDF$ representation).

we expect we could increase system scale by a factor of 10 while keeping all the virtual machines in the same network.

As indicated by our evaluation, the "weak link" in our system is the communication delay between the *Experiment Manager* and the rest of the modules. Aside from QoS solutions to ensure that this delay doesn't exceed predefined bounds, a possible alternative for demos at remote locations is to execute the *Experiment Manager* on the testbed, and only send graphical content to the remove location (e.g., via VNC or an equivalent method). The solution of a "portable testbed" deployed on the demo premises mentioned in Section 5.1 is another alternative, although this increases the deployment cost of the system.

6. CONCLUSIONS

In this paper we have presented an interactive experiment framework that makes it possible to dynamically control experiment scenarios, typically through direct user input. The proof-of-concept implementation of the framework used an *Experiment Manager* module to drive the entire system, the `dynamiQ` module for performing the actual network emulation, and a *Visualizer & User Interface* module to handle experiment visualization and user input. Some additional modules were used to retrieve the network topology from the running experiment, and to generate mobility traces for the experiment nodes.

`DynamiQ` receives control information through the interface with the *Experiment Manager*, then reconfigures the scenario, recomputes the communication conditions, and applies them into the experiment network. For the two latter purposes, `dynamiQ` leverages the QOMET network emulation set of tools. All these make it possible to provide the dynamic network emulation functionality required by the interactive experiment framework.

Our evaluation shows that the time lag introduced through the use of the `dynamiQ` module is not significant. Thus, on our experiment infrastructure, the average communication and computation time gaps are of the order of milliseconds, and their standard deviation of the order of at most tens of milliseconds. Moreover, the probability for the total time lag to be less than 20 ms is approximately 99.9%. The results are deemed acceptable given that these time gaps are considerably smaller than the update time interval of 500 ms used in our emulation experiments.

The current implementation of the interactive experiment framework has already been used for several demonstrations, including at Interop Tokyo 2014 and a planned demonstration at TridentCom 2015. We envisage using these experiences to finalize the design and implementation of the framework, so as it can be made available to interested parties as an easily-deployable "package".

7. REFERENCES

[1] R. Beuran, J. Nakata, T. Okada, L. T. Nguyen, Y. Tan, and Y. Shinoda. A Multi-purpose Wireless Network Emulator: QOMET. In *Proceedings of the 22nd IEEE International Conference on Advanced Information Networking and Applications (AINA 2008) Workshops, FINA 2008 symposium*, pages 223–228, 2008.

[2] R. Beuran, L. T. Nguyen, T. Miyachi, J. Nakata, K. Chinen, Y. Tan, and Y. Shinoda. QOMB: A Wireless Network Emulation Testbed. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM 2009)*, 2009.

[3] M. Carbone and L. Rizzo. Dummynet Revisited. *ACM SIGCOMM Computer Communication Review*, 40(2):12–20, 2010.

[4] D. Crockford. IETF RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON) . In *Internet Requests for Comments, Internet Engineering Task Force*, 2006.

[5] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE Simulator for DTN Protocol Evaluation. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools '09)*, 2009.

[6] T. Miyachi, K. Chinen, and Y. Shinoda. StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software. In *Proceedings of the Intl. Conf. on Performance Evaluation Methodologies and Tools (Valuetools 2006)*, 2006.

[7] OLSR.org. olsrd: an ad hoc wireless mesh routing daemon. `http://www.olsr.org/`.

[8] S. Zaitsev. jsmn: a minimalistic JSON parser in C. `http://zserge.com/jsmn.html`.