DNEmu: Design and Implementation of Distributed Network Emulation for Smooth Experimentation Control

Hajime Tazaki[1] and Hitoshi Asaeda[2]

[1] National Institute of Information and Communications Technology (NICT), Japan
[2] Keio University, Japan

Abstract. Conducting a realistic network experiment involving globally distributed physical nodes under heterogeneous environment introduces a requirement of experimentation control between the real world network and emulated/simulated networks. However, there is a gap between them to deploy network experiments. In this paper, we propose the *Distributed Network Emulator* (DNEmu) to fill the gap for the requirements of a planetary-scale network experiment. DNEmu addresses the issue of real-time execution with message synchronization through distributed processes, and enables us to evaluate protocols with actual background traffic using a fully controlled distributed environment. Through evaluation with micro-benchmarks, we find that our DNEmu prototype implementation is similar in terms of packet delivery delay and throughput to the existing non-virtualized environment. We also present a use-case of our proposed architecture for a large distributed virtual machine service in a simple control scenario involving actual background traffic on the global Internet. DNEmu will contribute to research in protocol evaluation and operation in a huge network experiment without interfering with the existing infrastructure.

Key words: distributed emulation, real-time simulation, ns-3

1 Introduction

Designing novel network protocols aiming to replace the current Internet architecture (i.e., clean-slate designs [6]) requires showing feasibility on the existing network before replacing it. Alternatively, analyzing the vulnerability of current protocols or unrevealed incidents (such as an outage for the popular site YouTube caused by inappropriate route announcement from Pakistan Telecom [1]) is important to avoid the accident happening in the future. As the demand for methods of studying communication is growing, a variety of network environments is necessary to be produced for the purpose of evaluating current and future protocols or architectures.

Network simulators greatly facilitate various network experiments. Researchers can create experimental scenarios and evaluate network architectures or protocols on simulated networks, without preparing large-scale and global wide net-

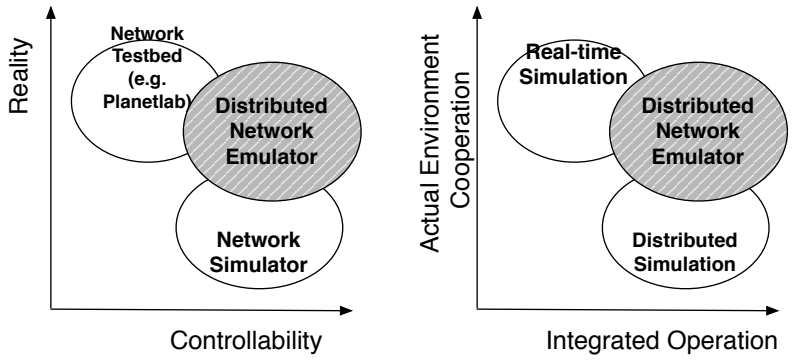


Fig. 1. Our work aims to fill the gap between network testbed and network simulator (left), and the gap between real-time simulation and distributed simulation (right). Both gaps are caused by lack of *smooth experimentation control*.

work infrastructures or complex communication environments such as mobile and wireless network experiments. However, as network simulators work with pre-defined communication models, it is impossible to evaluate protocols with unpredictable active traffic that occurs in the actual global Internet. This condition often precludes realistic evaluation and reduces the evaluation quality.

As an alternative solution, a planetary-scale network testbed such as PlanetLab [15] has recently been used for experimenting with active traffic on the global Internet. The scale of the number of nodes in the experiment, however, is limited to the number of physical nodes in the testbed. Moreover, the cost of the experiment controlling multiple nodes is regrettably high, especially when the number of nodes and complexity of the experimental scenario in the experiment grow.

More realistic experimental results have been gained by using ns-3, which is a novel network simulator [9]. Thanks to numerous contributions by researchers, ns-3 now has capabilities such as the ability to import actual traffic with a real-time scheduler, parallel distributed network simulation by Message Passing Interface (MPI) [7] for synchronized operation under a distributed environment, and Direct Code Execution (DCE) [11] to execute active running code in the simulation. Such functionalities help researchers conduct a realistic experiment for the evaluation of network protocols in a single toolset of software. However, it still lacks the key functionality for satisfying the requirements: the implementation of current distributed simulation in ns-3 cannot carry out simultaneous execution with the real-time simulation. It was impossible to perform an integrated operation by distributed simulation and cooperate with external traffic by real-time simulation simultaneously.

By considering the current picture of network experiments without *smooth experimentation control* as illustrated in Figure 1, our motivation for this work is filling the gaps in order to conduct planetary-scale network experiments. We introduce a novel distributed, real-time network emulation architecture, called

DNEmu (Distributed Network Emulator), to satisfy the following requirements for the network experiment.

- R1: The experiment should incorporate live traffic with the simulated background traffic.
- R2: We should have fully control of the experiment in a distributed environment.

Our design choice for DNEmu involves exploiting the existing MPI-based network simulation of `ns-3`, while considering the issue of real-time execution involving live traffic.

The contributions of this paper are two-fold. First, we have designed a distributed real-time emulation on a novel network simulator, ns-3, to achieve a global-scale network experiment with easier operation in a single toolset of the software. In the design phase of this study, we identified the issue of distributed simulation in real-time execution. Second, we have implemented a prototype of our proposed design and evaluated the similarity of basic network performance between the proposed architecture and a non-virtualized environment. To the best of our knowledge, no such architecture exists based on the combination of these two distinct simulation algorithms.

2 Background

This section briefly introduces two important functionalities of network simulation: *real-time simulation* and *distributed simulation*. Both help users of network simulators conduct network experiments in the distributed environment, but they are mutually exclusive, which motivates the pursuit of a solution in this paper.

2.1 Real-time Simulation

In contrast to traditional discrete event simulation, simulation synchronized with wallclock time of a computation node (called real-time simulation) was proposed by Fall [5] and implemented on the ns-2 network simulator, which is the primary network simulator of ns-3. As depicted in Figure 2, instead of conventional discrete event scheduling, *real-time scheduling* allows us to wait for the next pending event until wallclock time corresponds to time in the simulator, and thus the clock inside the simulation is synchronized with the outside the simulator, allowing communication. By using this enhanced scheduler, the application in the simulator is able to include live traffic from the outside (e.g., the Internet).

While real-time scheduling provides external interoperability to the network simulator, it does not schedule events in real time. When a large number of events attempt to execute at the same time t, the next pending event will be delayed and the clock will then be desynchronized. Performance improvement in the simulation core will help us to ensure that the event scheduling meets these deadlines [13].

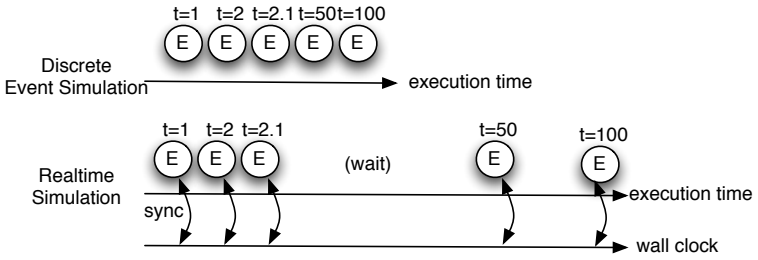


Fig. 2. Real-time simulation. The event at simulation time t waits until the wallclock time reaches t.

2.2 Parallel and Distributed Simulation

Parallel and distributed network simulation [18] has been studied with the requirement of rapid execution of complex and large-scale network simulation. By utilizing multiple logical processes (LPs) distributed to multiple nodes, the simulation has accelerated execution time, while ensuring the same results with a synchronized clock among the distributed processors.

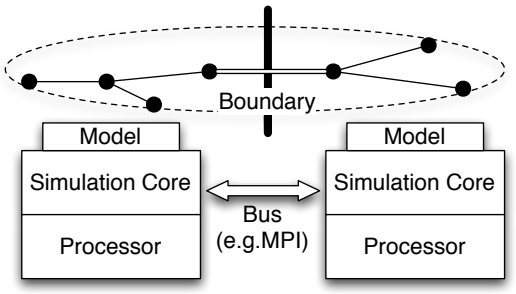


Fig. 3. Parallel and distributed simulation. Using several processors or physical nodes distributes the load of simulation processes and accelerates execution with coordination of time synchronization.

As shown in Figure 3, the topology on the simulated model could be shared among the different processors or physical nodes. When the message (or event) goes across the simulation boundary in the shared topology, a time-stamped message is exchanged via the message bus between the different simulation cores and processed at the second processor. The result then goes back to the original simulation core without adding to the original processor's workload and the final execution time of the simulation will be thereby shortened.

Though the previous studies in the distributed simulation have focused on the performance improvement of the network simulation, it is also necessary to study the environmental synchronization among the distributed nodes, for an integrated network experiment with simple operation.

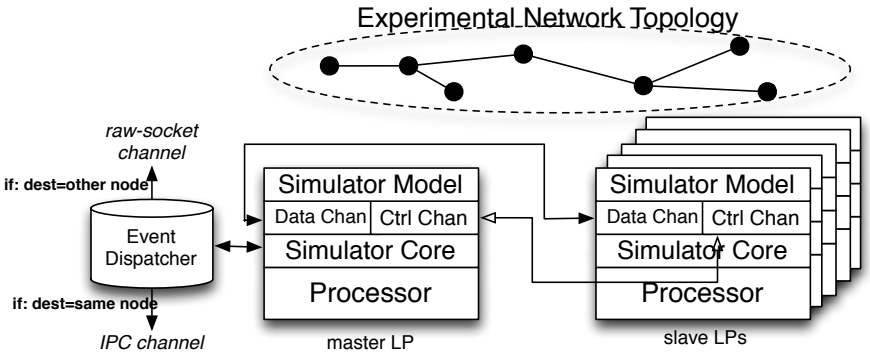


Fig. 4. The DNEmu architecture for distributed real-time emulation.

2.3 What is Missing?

Although the above two simulation algorithms are distinct, both of these algorithms will contribute to the execution of a network experiment at distributed locations. Controlling a complex experimental topology in a single scenario script allows for easier deployment of the network experiment into hundreds of nodes, and actual traffic generated by actual applications improves the quality of the simulation.

However, the current implementation of the distributed simulation in `ns-3` does not allow external traffic to be imported via a real-time scheduler. Further, designing additional functionality for real-time execution in the distributed simulation is also not straightforward, because of the timescale differences within the simulation. The following sections detail a framework for filling the gap of the current simulation architecture.

3 The Architecture

In this section, we present our proposed architecture, DNEmu for distributed real-time network emulation satisfying the above requirements.

3.1 Overview

As stated in Section 1, the goal of this paper is to provide a network experiment framework, which both involves live traffic, and also uses a distributed environment.

Our design choice to accomplish this goal is to extend existing frameworks of distributed simulation and real-time simulation, and be able to utilize them simultaneously. Current distributed simulations require time synchronization between distributed processors (or systems), but our proposed architecture does not require any such synchronization, since the time should "walk along" with

the wallclock time of each system in the real-time scheduler (details are discussed in Section 3.3).

In addition, we do not use MPI as an *inter-connect* for data transfer between logical processes. Instead, we use the usual *raw-socket* based communication via the usual network interface cards (or tunnels in some physical network environment cases). We do this because the buffering strategy of MPI for optimized message synchronization does not allow the exchange of real-time messages between the LPs (details are discussed in Section 3.3). This separation of control channel and data channel allows us to use a distributed real-time scheduler in a network simulator.

Figure 4 shows the overview of the DNEmu architecture. While MPI is used as a control channel between distributed logical processes, the real-time scheduler handles external traffic coming from outside the simulator, and also appropriately schedules the traffic on the data channel at the simulation boundary (i.e., the link between logical processes). A simulation scenario among the distributed nodes can be shared, and execution is handled only at the master node and synchronized with all of the distributed slave nodes.

3.2 Principles

During the design phase of our architecture, the following principles for design choice, from several directions, were considered.

Less dependency on hardware environment: The architecture and its implementation should not be restricted to any particular hardware environment (e.g., dedicated cluster computers for distributed simulation) in order to allow distribution all over the world. It should use standard hardware with a common operating system.

Less dependency on external toolsets: Dependency on external toolsets is minimized to reduce the complexity of the operation of network experiments. Since the architecture is a framework for any network experiments, and could be utilized in many different ways, it should be able to tolerate for future extensions of the architecture.

3.3 Functional Components

This section explains the functional components required for the DNEmu architecture.

Event dispatcher: We introduce a new simulated event dispatcher for distributed real-time emulation. Existing distributed simulation transfers events (i.e., packets) to the other endpoints via the MPI library if the events cross the simulation boundary. Instead, in our architecture, the events are translated and transferred to an emulated *raw-socket* (or tap device), and delivered to the other endpoints over a connected network when distributed emulation is defined in the simulation scenario. If the destination node of an event transfer is the same as the original node, it conducts inter-process communication (IPC) for the event transfer instead of *raw-socket* based communication.

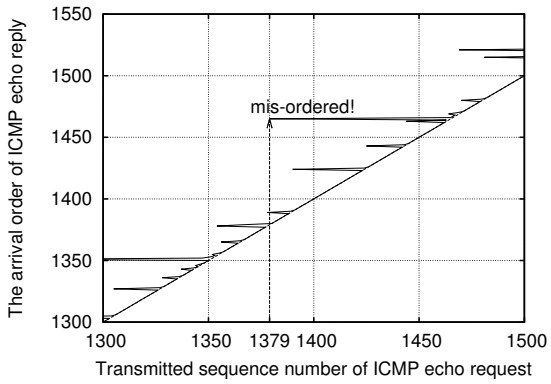
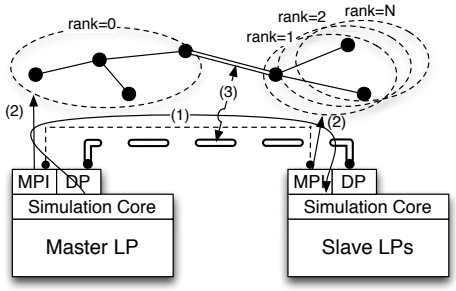


Fig. 5. The order of the arrival of echo reply packets at Tx (in Figure 7). The protruded point presents mis-ordered arrival of echo reply packets. The buffering effect of the MPI library reveals mis-ordered ping results.

Message passing between LPs: In addition, the function of the message passing in a time-stamped order at the boundary of LPs is removed since we do not have to synchronize the simulation clocks, as the real-time scheduler uses wallclock instead. However, we still use the MPI library, because it easily enables to control distributed nodes by providing the initial rendezvous, bootstrapping, and to assign of MPI *rank*, which is the unique number of each logical process. We then utilize existing raw-socket based communication at the simulation boundary for data transfer between simulated nodes. During the implementation of our prototype, we found that using MPI as a data channel of the *inter-connect* is not suitable since the buffering behavior of the MPI message transfer prevents real-time traffic delivery to another node. For example, the order of arrival of ICMP echo request packets at a destination is different from the order of transmitted packets. Figure 5 shows the order of arrival of ICMP Echo reply packets in function of the transmitted sequence number from our prototype implementation based on an MPI-based *inter-connects*. We can see that the packet transmitted with sequence number 1379 was echoed backed in 1465th place. Such a non-negligible effect for real-time simulation should be removed. One possible solution to this issue is using a high-performance cluster computer with remote direct memory access (RDMA) to reduce the delay on the *inter-connect*. However, such hardware environments are not commonly available and restrict the use of our architecture. We therefore do not use it in our design principles.

Figure 6 depicts the bootstrap sequence of distributed real-time emulation with a pseudo-scenario script. When the simulation is executed at a node, the node is the master LP ($rank = 0$), others are slave LPs, then:

(1) The MPI handles the execution of the program at remote (i.e., slave LPs), and assigns *rank* using a static configuration for the MPI executable as usual.
(2) Decide the behavior of each simulated node based on their assigned *ranks* (e.g., each AS (Autonomous System) setup in the network).




```
 1: main () {
 2:  MPI_Init ();                  // (1)
 3:  (use of Real-time scheduler)
 4:  if (rank == 0) {              // master
 5:     (configuration of AS1)     // (2)
 6:     (establish inter-connect) // (3)
 7:  }
 8:  else {                        // slaves
 9:     (configuration of ASN)     // (2)
10:     (establish inter-connect) // (3)
11:  }
12:  return;
13: }
```

Fig. 6. The sequence of bootstrapping using DNEmu.

(3) Configure the *inter-connect* link between simulation boundaries at each LP.

When a packet in the simulation reaches the boundary node of each LP, and is going to forward to another LP, this *inter-connect* is used for the data channel between distributed emulation. The sequence of (1) and (2) is the same as in the standard distributed simulation, while (3) is introduced by our DNEmu architecture.

In the architecture of DNEmu, the separation of the control and data channels allows the real-time execution of the distributed simulation. While the MPI only takes care of control messages for distributed logical processes as a *control channel*, raw-socket based *inter-connects* work as a *data channel*. Such a design choice fulfills our requirement of distributed real-time emulation.

4 Evaluation

In this section, based on our prototype implementation of DNEmu on `ns-3`, we present a performance measurement using micro-benchmarks on our proposed architecture, with the Linux container-based CORE (Common Open Research Emulator) distributed network emulator [2] as an alternative. The objectives of this evaluation are to show the proof of the concept of DNEmu and the similarity of our approach in terms of packet delivery with alternative network emulators. We then give a possible use-case of DNEmu over globally distributed virtual machines.

4.1 Setup

All of our experiments were conducted on two Linux systems, Node A and B as shown in Figure 7, which were equipped with an Intel Core i7-2600 (3.4GHz) and an AMD Opteron 6128 (2.0 GHz) processor. Two distributed nodes were located in Tokyo and Kanagawa in Japan respectively, with nine hops between them. Both systems ran with Ubuntu 10.04 64-bit with kernel versions 2.6.32.29 and

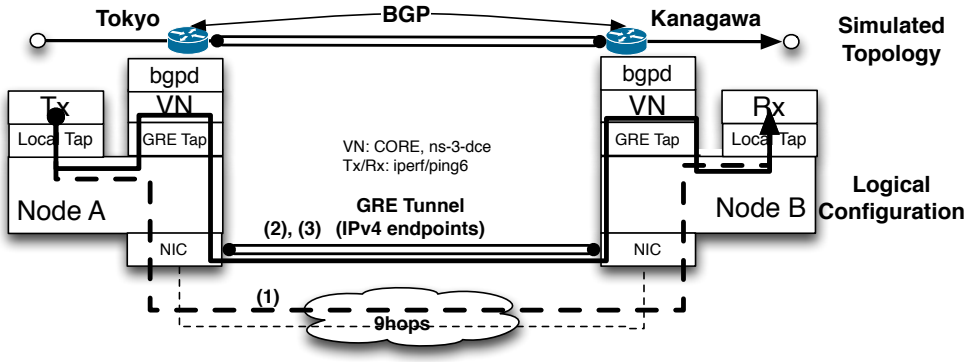


Fig. 7. Experimental setup of micro-benchmark.

2.6.39.4. We then installed the modified version of ns-3-dce-quagga-umip[1] with our distributed real-time emulation extension. We also used a CORE network emulator version 4.2svn2 (20110919) on both systems.

By using two distributed nodes, we configured the experimental network as illustrated in Figure 7. In the case of `ns-3`, we set up one simulated node in each distributed physical node and configured Zebra BGP routing, based on Direct Code Execution (DCE) [11], to exchange their owned route information at the virtual node (VN), as shown in Figure 7. The forwarding processes in `ns-3` were also configured with the modified version of DCE, Linux kernel integration (`i.e., ns-3-linux`[2]), and `DlmLoader`. Each simulated node was connected via a `tun/tap` interface (i.e., Local Tap) configured in the underlying operating system in order to inject traffic from outside the simulation. The simulated node was also configured with a GRE tunnel via a `gretap` interface (i.e., GRE Tap) for the *inter-connection* of distributed nodes. The entire configuration is described in a single simulation script of `ns-3` with DNEmu.

In this experimental network, we used `ping6` and `iperf` to measure the experimental traffic. All traffic was generated by Node A's underlying operating system (i.e., Tx), directed to the VN via Local Tap, and forwarded to Node B's VN according to the routing information exchanged by BGP, then delivered to Node B's underlying operating system (i.e., Rx) via Local Tap.

In the case of CORE network emulation, we also configured the same network topology and VN as configured in `ns-3` above.

4.2 Micro-benchmark

By using the previous setup, we conducted round-trip time (RTT) and available throughput measurements. We first measured the performance of direct commu-

[1] Original version is downloaded from `http://code.nsnam.org/thehajime/ns-3-dce-quagga-umip/`

[2] Original version is downloaded from `http://code.nsnam.org/mathieu/ns-3-linux/`

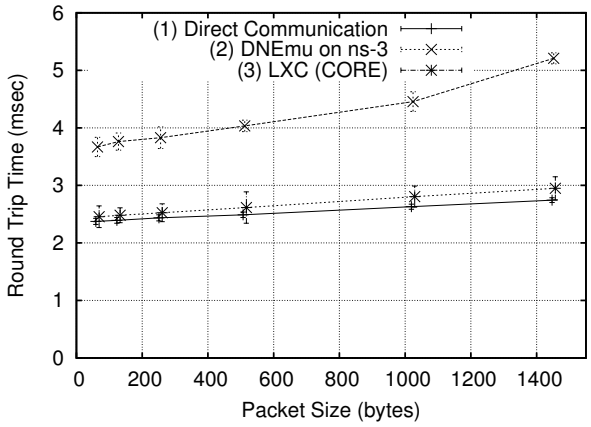


Fig. 8. Packet delivery delay in function of packet size with standard deviation, from 5000 replications.

nication between Node A and B (as shown in (1) in Figure 7) without involving any virtualized node, in order to show the performance baseline of the network environment. We then measured the performance via our DNEmu architecture (2) and the CORE network emulator (3).

Figure 8 represents the RTT between two endpoints with an interval of 10 milliseconds in function of the size of packets (64, 128, 256, 512, 1024, and 1452 bytes) with standard deviation, from 5000 repetitions of the `ping6` command. As shown in the result of direct communication, the base RTT in this network environment in this experiment was around 2.5 milliseconds. The CORE emulator based on Linux Containers (LXCs) scored with an additional $0.1-0.2$ millisecond delay on which to the base RTT. This minimum amount of overhead is achieved by lightweight virtualization of LXC. On the other hand, `ns-3` with distributed real-time emulation added considerable delay to the base performance: almost twice as much as direct communication in the case of 1452 bytes packet.

Figure 9 also represents the result of measuring bandwidth by the `iperf` command using the TCP Reno algorithm between two endpoints. We recorded the bandwidth every second for 10 times and plotted the standard deviation from repetitions.

We can see the performance disadvantage of DNEmu in delivery delay and throughput. This comes from the packet processing delay inside `ns-3` via *raw-sockets* and *tap-devices*. Performance improvement of such functionality is required to obtain accurate results although it is beyond the scope of this paper. However, the trend of packet size growth is similar to that in direct communication and other emulators (i.e., CORE).

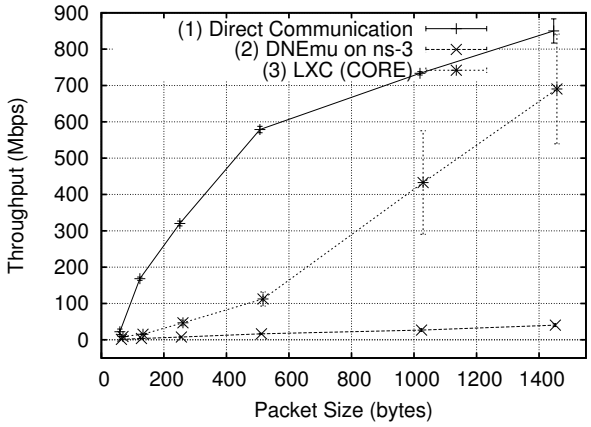


Fig. 9. Throughput (TCP) in function of packet size.

4.3 Use-Case: Experiment using a Globally Distributed Virtual Machine Service

One possible use-case of DNEmu is that of network experiments in a distributed network testbed based on a virtual machine service provider. Due to the limitation of the number of nodes in the network experiment with such a virtual node, a network simulator contributes to increasing the number of experimental nodes on the testbed with a simple and controllable scenario description. This section presents a NEtwork MObility (NEMO) handoff experiment involving distributed located home agents maintained by BGP and OSPFv3.

Figure 10 depicts our experimental network configuration for this use case using three different sites. MR is a mobile router operated with NEMO software, MNN is a mobile network node moving with the MR, BS0 and BS1 are base stations equipped with IPv6 router functionality, HA is a home agent, ARs are access routers bridging three distributed networks via a tunnel, and CN is a correspondent node. Access routers, which are located at the boundary of each site, operate using the BGP routing protocol of Zebra to exchange route information in the network. In addition to the experiment shown in Section 4.2, we used a globally distributed virtual machine operated by the WIDE cloud service, which is organized by the WIDE project[3] and composed of Kernel-based virtual machines (KVM) [10] located at nine distributed sites. We used a single KVM node located in San Francisco and configured with a GRE tunnel and an IP6-in-IP6 tunnel as *inter-connects* between each distributed sites.

[3] `http://www.wide.ad.jp`

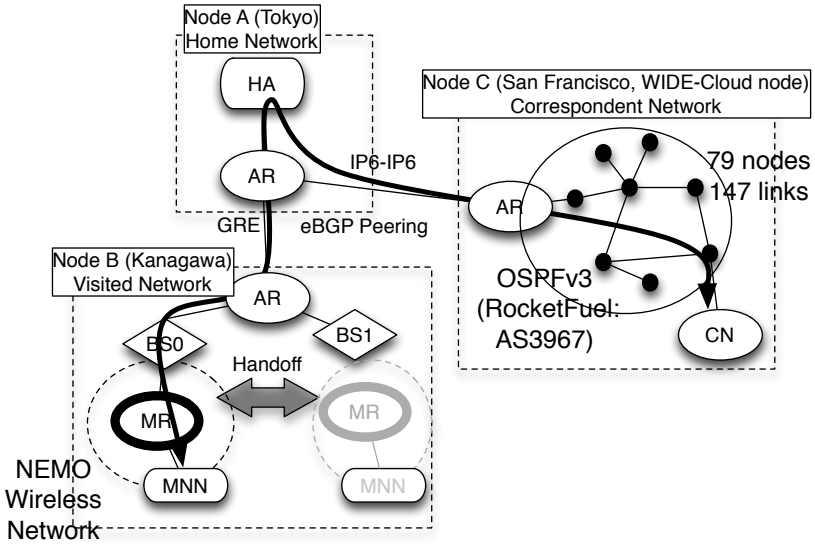


Fig. 10. Experimental use-case of an all IPv6 mobile network with a WIDE Cloud (VM cloud) node as a complex network configuration with a single controlled emulation scenario script.

In node A, the HA was operated with an Usagi-patched Mobile IPv6 implementation (UMIP)[4], and Linux kernel[5] to provide mobile networks for the MR. In node B, two base stations served Wi-Fi access points to the MR and created different wireless cells. The MR switched its point of attachment to the base station and obtained a different network prefixes to change its Care-of-Address, while the MNN followed and used the same address served by the upper MR. In node C, the reachability to the CN was managed by the OSPFv3 protocol executed by the Zebra `ospf6d` daemon using `ns-3` DCE functionality. The operated network was created by the RocketFuel topology dataset [19] and we used an Exodus (AS3967) database, which consists of 79 nodes and 147 links.

During this experiment, the MR and the MNN moved around between BS0 and BS1 with re-registration of binding to its home agent, and the MNN continuously sent ICMPv6 echo requests to the peer node (CN in Figure 10) to measure the duration of the disrupted communication during a handoff.

All of the above experimental scenarios were configured in a single `ns-3` script and controlled at the master node (node A) with our DNEmu architecture.

Figure 11 shows the result of handoff during this experiment. In this figure, the MR switched its point of attachment from BS0 to BS1 involving the handoff procedure with the HA, creating nine-second disruption of the `ping6` command.

[4] USAGI-patched Mobile IPv6 for Linux: `http://umip.linux-ipv6.org/`, downloaded Jul 7 2010 version.

[5] `http://git.kernel.org/?p=linux/kernel/git/davem/net-next-2.6.git`, downloaded Aug 19 2010 version.

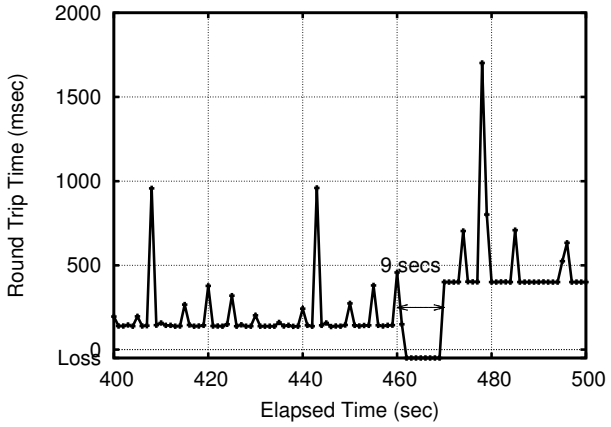


Fig. 11. Round trip time value between MNN and CN in function of elapsed time of the experiment. The RTT value "Loss" represents the disruption of communication caused by changing the point of attachment at the MR.

At around 460 seconds, the MR changed the IP address of the egress interface of the mobile router, re-registered the binding information to the home agent, and updated the bidirectional tunnel between the MR and HA. After that, the `ping6` command continuously recovered without changing the MNN's address insured by NEMO. We can see longer RTT after the handoff since BS1 adds a 100-millisecond delay in its simulation scenario. The fluctuating RTT is also seen because of path characteristics between Tokyo and San Francisco.

Such an experiment could be executed without involving existing environmental modifications. Neither network configurations (IP address and prefix allocation) nor operating system and kernel modifications are required.

5 Discussion

This section discusses the results of the above evaluation and future directions for this work.

From an architectural design point of view, the proposed architecture is not a *simulator*, because there is no reproducibility of the experimental result: time synchronization is not performed among the distributed nodes, un-modeled behavior at wallclock based timescales always produces different result, and background traffic at the *inter-connect* always changes. However, the fully integrated control via a network simulator greatly helps the easier operation of network experiments with our DNEmu architecture. We therefore focused on a network simulator as a toolset of network experiment.

Moreover, the architecture only relies on common features of the current operating systems and does not require upgrading the system, which sometimes would be a constraint in network experiments (e.g., the case of PlanetLab [3]).

Such a design choice will help with global-scale deployment without the obstacle of software dependency.

The performance of the real-time scheduler plays an important role in this architecture. If events are not finished before the deadline of the wallclock time, the clock will be desynchronized, as mentioned in Section 2.1. Performance improvement in the simulation core is required, and would be a future direction of this work.

6 Related Work

The architecture proposed in this paper is related to two areas of work: integrated emulators and external testbed controllers. This section highlights the relations with the existing work.

Integrated emulators here refers to the experimental tool that is able to emulate various environments from operating systems (by virtualization) to network conditions (by, for example, Wi-Fi emulation on a wired link). This enables researchers to experiment in advance with their proposed protocols, architectures, or operations without wholesale deployment of the environment. CORE (Common Open Research Emulator) [2] exploits the lightweight virtualization technology of the operating system (i.e., IMUNES [16], netns [4]) and allows us to execute existing applications over emulated links. This tool supports describing the arbitrary topology with a GUI and executing on a distributed environment from a single controller. While CORE is able to automate the flow of a network experiment in a realistic environment, it requires the operating system's support to execute it. By contrast, the architecture proposed in this paper virtualizes everything in the user-land application based on `ns-3` and requires no extension to the kernel. PrimoGENI [20] and ROSENET [8] are both integrated network simulators with an emulation facility via an emulation gateway, and achieving flexible and scalable network experiments. However, due to the number of components involved in an experiment, the potential complexity will increase. Our DNEmu architecture achieves distributed emulation involving only with a single toolset of the software and can control everything in a single simulation scenario. This is the strength of our architecture compared to alternatives.

An external testbed controller is a tool (or toolset) to help the execution of network experiments allowing easy operation and repeatability of the experiment. OMF (cOntrol and Management Framework) [17] is a toolset of the software to control experiments, manage the experimental component, and conduct measurement via a unified controller. These have been successfully deployed into an ORBIT testbed [14] to operate on hundreds of physical nodes. NEPI (Network Experiment Programming Interface) [12] has been proposed as a general framework for network experiments with a `python` programming interface. It can operate across multiple types of network testbeds such as PlanetLab, EmuLab [21], or ORBIT etc. While these testbed controllers provide an abstract model of the network experiments, our DNEmu architecture only targets at a

specific network simulator (i.e., `ns-3`) in order to reduce external software dependencies of the toolset.

7 Conclusion

In this paper, we have designed the DNEmu architecture based on the combination of a distributed simulation and real-time simulation. To the best of our knowledge, there is no other such architecture based on the combination of these two distinct scheduling algorithms. The architecture has satisfied our requirements for a network experiment within a single toolset of software. Through the prototype implementation of our architecture, micro-benchmarking has shown similar trends with packet delivery delay and throughput, and a use-case has been presented on top of a globally distributed cloud service with a distributed synchronized network experiment. Yet the implementation is at an early stage and we have already found some performance drawbacks. However, our architecture will benefit the user who is going to deploy network experiments with globally distributed nodes, without being concerned about a complex toolset for the experiment.

The contributions of this paper are two-fold. First, we have designed a distributed real-time emulation on a novel network simulator `ns-3` to achieve a global-scale network experiment with easier operation in a single toolset of the software. Second, we have implemented a prototype of DNEmu and evaluated the similarity of basic network performance between our proposed architecture and a non-virtualized environment.

Acknowledgment

The authors wish to thank the WIDE Cloud Computing Working Group of WIDE project for their support of our experiment. We also thank Clare Horsman for her comments to improve the paper.

References

1. Pakistan hijacks YouTube. `http://www.renesys.com/blog/2008/02/pakistan-hijacks-youtube-1.shtml` (accessed 2011-10-17).
2. J. Ahrenholz, C. Danilov, T. Henderson, and J. Kim. CORE: A real-time network emulator. *Proceedings of Military Communications Conference*, pages 1 –7, Nov. 2008.
3. S. Bhatia, G. Di Stasi, T. Haddow, A. Bavier, S. Muir, and L. Peterson. Vsys: a programmable sudo. In *Proceedings of the 2011 USENIX annual technical conference*, USENIX ATC'11, Berkeley, CA, USA, 2011. USENIX Association.
4. S. Bhattiprolu, E. W. Biederman, S. Hallyn, and D. Lezcano. Virtual servers and checkpoint/restart in mainstream linux. *SIGOPS Oper. Syst. Rev.*, 42(5):104–113, 2008.

5. K. Fall. Network emulation in the VINT/NS simulator. In *Proceedings of International Symposium on Computers and Communications*, pages 244–250. IEEE, July 1999.
6. A. Feldmann. Internet clean-slate design: what and why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64, July 2007.
7. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.
8. Y. Gu and R. Fujimoto. Applying parallel and distributed simulation to remote network emulation. In *Proceedings of the Winter Simulation Conference*, WSC '07, pages 1328 –1336, Dec. 2007.
9. T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *Proceedings of the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06. ACM, 2006.
10. A. Kivity. Kernel Based Virtual Machine. `http://www.linux-kvm.org/` (accessed 2010-12-17).
11. M. Lacage. *Experimentation Tools for Networking Research*. PhD thesis, UNIVERSITE DE NICE-SOPHIA ANTIPOLIS, 2010.
12. M. Lacage, M. Ferrari, M. Hansen, T. Turletti, and W. Dabbous. NEPI: using independent simulators, emulators, and testbeds for easy experimentation. *ACM SIGOPS Operating Systems Review*, 43(4):60–65, Jan. 2010.
13. D. Mahrenholz and S. Ivanov. Real-Time Network Emulation with ns-2. In *Proceedings of International Symposium on the Distributed Simulation and Real-Time Applications*, DS-RT 2004, pages 29–36, Oct. 2004.
14. M. Ott, I. Seskar, R. Siraccusa, and M. Singh. ORBIT testbed software architecture: supporting experiments as a service. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, TridentCom 2005, pages 136–145, Feb. 2005.
15. L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.
16. Z. Puljiz and M. Mikuc. IMUNES Based Distributed Network Emulator. *Proceedings of the International Conference on Software in Telecommunications and Computer Networks*, pages 198 –203, Oct. 2006.
17. T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar. OMF: a control and management framework for networking testbeds. *SIGOPS Oper. Syst. Rev.*, 43:54–59, Jan. 2010.
18. G. Riley, R. Fujimoto, and M. Ammar. A generic framework for parallelization of network simulations. In *Proceedings of the 7th International Symposium on Modeling Analysis and Simulation of Computer and Telecommunication Systems*, pages 128–135, Oct. 1999.
19. N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP topologies with rocketfuel. *IEEE/ACM Transactions on Networking (TON)*, 12(1):2–16, Feb. 2004.
20. N. Van Vorst, M. Erazo, and J. Liu. PrimoGENI: Integrating Real-Time Network Simulation and Emulation in GENI. In *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation*, PADS 2011, pages 1–9. IEEE, June 2011.
21. B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of the 5th symposium on Operating systems design and implementation*, OSDI '02, pages 255–270, 2002.