MSWiM Demo Abstract: Direct Code Execution: Increase Simulation Realism using Unmodified Real Implementations

Hajime Tazaki*, Emilio Mancini°, Daniel Câmara°, Thierry Turletti°, Walid Dabbous°

*University of Tokyo, Japan °INRIA, France

Abstract

We propose to demonstrate Direct Code Execution (DCE)[1], a ns-3 simulation framework that enables reproducible network experiments using real Linux kernel space protocol stacks along with POSIX socket based protocol implementations. In addition to increased experimentation realism, it offers a highly configurable topology environment and allows easy debugging of communication protocols distributed other multiple nodes. Our demonstration will showcase two typical use cases of DCE: information-centric networking over mobile ad hoc network using the PARC CCNx code, and a seamless handoff experiment based on a Linux Multipath TCP (MP-TCP) implementation.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems

Keywords: Network Stack; Experimentation; Software Development; Direct Code Execution; Linux

1. INTRODUCTION

The increasing demand for reproducible network experiments requires sophisticated tools to conduct arbitrary network experiments at large scale. Indeed, many features are needed in addition to providing experimentation realism such as the possibility to configure the network topology in a flexible way, to easily replicate experiments at low cost, to run experiments at large scale and to easily debug network protocols even over distributed nodes. Container-based emulation (CBE) [3] approaches such as Mininet provide experimentation realism and easy/low cost replication but they are limited by the physical resources of the emulation machine and do not provide easy debugging facilities. Shared testbeds such as PlanetLab [9] offer experimentation realism but the performance obtained highly depend on network conditions which are not controlled, and configurability of the topology is limited. As for network simulators, they provide a fully reproducible network environment, with highly configurable topology but they usually lack of functional realism because most often they use simplistic models of network protocol implementations.

Table 1: Requirements for reproducible network experiments.

	Simulators	Testbeds	Emulators
Functional Realism	???	✓	✓
Timing Realism	✓	✓	✓
Topology Flexibility	✓	(limited)	✓
Easy Replication	✓	✓	✓
Easy Debug	✓		
Scalability	✓		

Porting existing network protocol implementations to network simulators is one possible option to increase the level of realism of experiments. OppBSD [2] or INETQuagga [1] take this approach to reuse existing protocol implementations (i.e., TCP/IP stack of FreeBSD and Quagga routing protocol suite), but they still require to manually patch for a particular network simulator, which is a complex task to perform. This makes it difficult to follow the evolutions and improvements of the simulator code. Network Simulation Cradle (NSC) [5] introduces a different technique; it automatically generates C source files of different operating system's network stacks (e.g., FreeBSD, Linux, OpenBSD, lwip), and builds shared libraries used in the network simulators. The automation alleviates the cost of tracking the latest version of codes and supports a wide range of existing code on a single framework, but additional effort is still required to introduce arbitrary protocol implementations in addition to TCP.

In this demo we showcase Direct Code Execution (DCE), the first open source framework integrating both Linux kernel space protocol stack and POSIX socket based user space application code within a discrete-event network simulator. DCE uses the traditional library operating system (LibOS) approach as Exokernel [6] in its core architectural design to enable running and evaluating real network proto-

[1]Project web page: `http://www.nsnam.org/overview/projects/direct-code-execution/`

MobiWac'13, November 3–8, 2013, Barcelona, Spain.

http://dx.doi.org/10.1145/2508222.2512837.

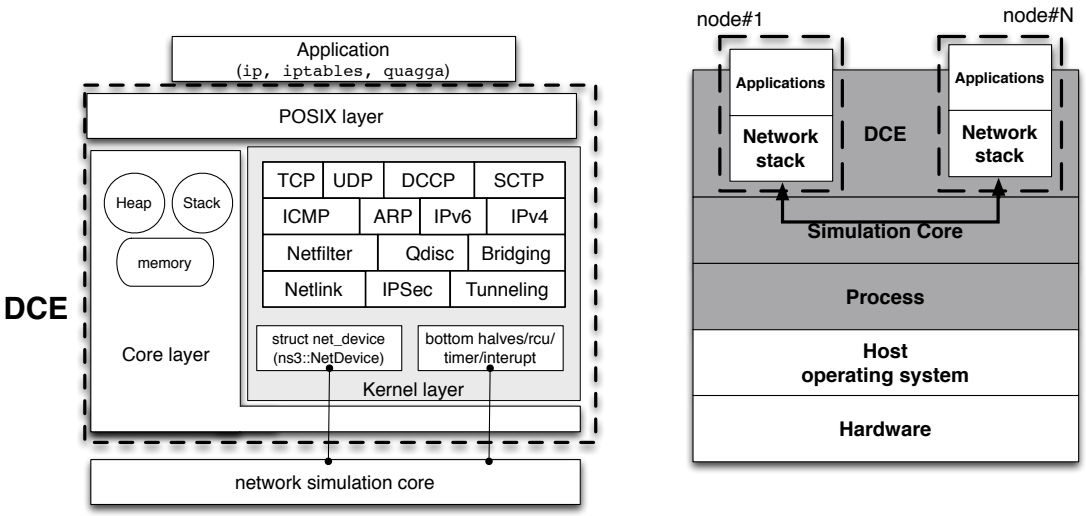

Figure 1: Architecture of Direct Code Execution.

col implementations. As a result, DCE provides functional realism to network simulation-based experimentation, which traditional simulators do not have, as shown in Table 1.

2. DCE SYSTEM OVERVIEW

The design of DCE is structured around three separate components as depicted in Figure 1.

- **Core.** The lowest-level *core* module handles the virtualization of stacks, heaps, and global memory. It uses a *single-process model* that executes every simulated process within the same host process and carefully isolates the namespace of each simulated process.
- **Kernel layer.** The kernel layer takes advantage of the *core* services to provide an execution environment to the Linux network stack within the single-processed network simulator. The services of kernel such as the Linux *bottom halves*, Read-Copy-Update (RCU) [7], scheduler, and timer API are re-implemented as a new architecture based on the `asm-generic` implementation to minimize the modifications of the original kernel code.
- **POSIX layer.** The POSIX layer builds upon the *core* and *kernel* layers to re-implement the standard socket APIs used by emulated applications.

Theoretically, there is no limitation regarding to the kind of protocol or application that may run over DCE without any manual modifications to the original code. In practice, however, we may need to extend the *POSIX layer* to add missing system calls only if the new application requires. Today, DCE supports a broad range of existing implementations running on `ns-3`: `Linux kernel` (2.6.36, 3.4-3.10 versions), `quagga` (ripd, ripngd, ospfd, ospf6d, bgpd, and rtadv), `ccnx`, `iperf`, `ip`, `ping/ping6`, `umip`, `bind9`, `unbound`, `thttpd`, and bittorrent (`opentracker`, `rasterbar`).

3. DEMONSTRATION DETAIL

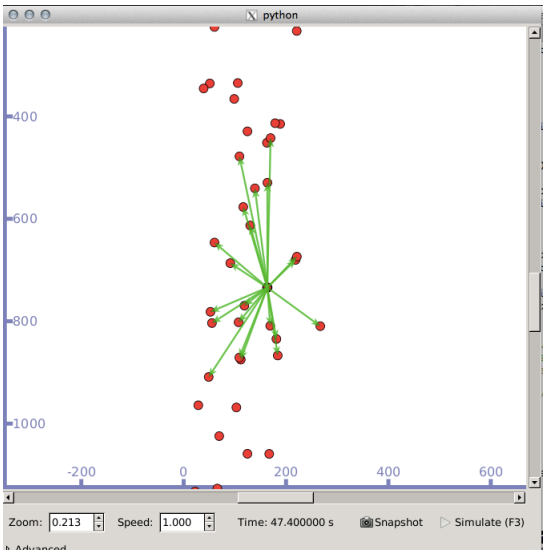

Figure 2: CCNx on WiFi mobile ad hoc network.

We plan to demonstrate two use cases to present major features of DCE: the first one involves the CCNx protocol implementation running in user space and the second one, the MPTCP Linux implementation running in kernel space.

User space protocol implementation running on DCE: CCNx[2] over MANET nodes

Content Centric Networking (CCN) [4] is a new paradigm proposed by PARC that transforms traditional host-to-host based communication into content-based communication. This paradigm is well suited to MA-NETs, because most application scenarios in mobile ad hoc networks are data-centric in nature [8].

[2]`http://www.ccnx.org/`

In this first use case, we will showcase the PARC CCN implementation, called CCNx, over a dynamic topology provided by `ns-3`.

Kernel space protocol stack running on DCE: MPTCP over LTE and WiFi links

Multipath TCP (MPTCP) [10] is an extension of the standard TCP allowing to use multiple subflows with different IP addresses without having to modify user space applications. Basically, this new transport protocol makes it possible to increase the throughput of an application by running it over multiple links, and it also enables transparent handoff using multiple IP addresses.

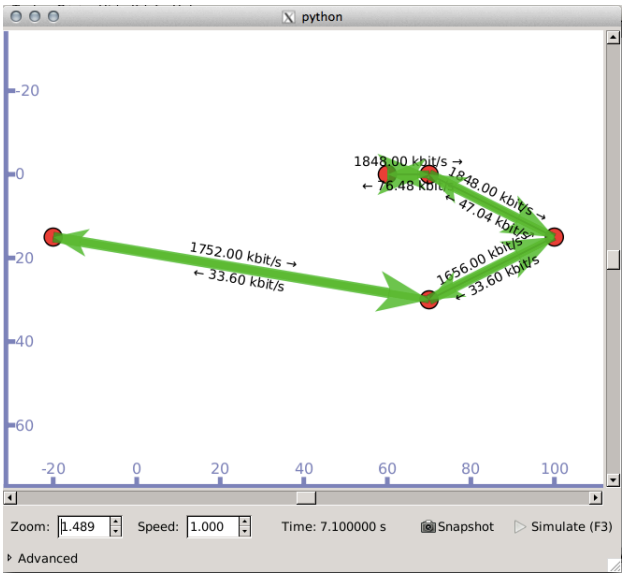

Figure 3: Handoff simulation using the Linux MPTCP implementation.

In this second use case, we will use a kernel space Linux MPTCP implementation[3] on DCE / `ns-3` and with the support of various user space applications (`quagga`, `ip` command, `udhcpd`, `iperf`). Multiple addresses will be provided to mobile nodes using two different wireless technologies supported by `ns-3`: LTE and WiFi. So, MPTCP will switch its primary address between the two types of links according to the node movement, trying to keep the ongoing TCP session available. Furthermore, the TCP session uses multiple subflows to increase the goodput if two wireless links are available.

Along with DCE / ns-3, our demonstrations will include animated visualization of simulated nodes, traffic status, as well as performance graphs obtained from each simulation.

Acknowledgments

This research was partially supported by INRIA and the Japanese Society for the Promotion of Science (JSPS) Joint Research Projects program in the context of the Simulbed associated team.

[3] `https://github.com/multipath-tcp/mptcp`

4. REFERENCES

[1] INETQuagga: OMNeT++ wiki. `http://www.omnetpp.org/pmwiki/index.php?n=Main.INETQuagga`. (Accessed July 1st 2013).

[2] Bless, R., and Doll, M. Integration of the freebsd tcp/ip-stack into the discrete event simulator omnet++. In *Simulation Conference, 2004. Proceedings of the 2004 Winter* (2004), vol. 2, pp. 1556–1561 vol.2.

[3] Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N. Reproducible network experiments using container based emulation. In *Proceedings of the 2012 ACM CoNEXT conference* (2012), CoNEXT '12.

[4] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies* (Dec. 2009), CoNEXT '09, ACM, pp. 1–12.

[5] Jansen, S., and McGregor, A. Simulation with real world network stacks. In *Proceedings of the 37th conference on Winter simulation* (2005), WSC '05, Winter Simulation Conference, pp. 2454–2463.

[6] Kaashoek, M. F., Engler, D. R., Ganger, G. R., Briceño, H. M., Hunt, R., Mazières, D., Pinckney, T., Grimm, R., Jannotti, J., and Mackenzie, K. Application performance and flexibility on exokernel systems. In *Proceedings of the sixteenth ACM symposium on Operating systems principles* (New York, NY, USA, 1997), SOSP '97, ACM, pp. 52–65.

[7] McKenney, P. E. *Exploiting Deferred Destruction: An Analysis of Read-Copy-Update Techniques in Operating System Kernels*. PhD thesis, Oregon State University, 2004.

[8] Meisel, M., Pappas, V., and Zhang, L. Ad hoc Networking via Named Data. In *Proceedings of the fifth ACM international workshop on Mobility in the evolving internet architecture* (2010), MobiArch '10, ACM, pp. 3–8.

[9] Peterson, L., Anderson, T., Culler, D., and Roscoe, T. A blueprint for introducing disruptive technology into the Internet. *SIGCOMM Comput. Commun. Rev. 33*, 1 (2003), 59–64.

[10] Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., and Handley, M. How hard can it be? designing and implementing a deployable multipath tcp. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 29–29.