Designing a Testbed for Large-scale Distributed Systems

Christof Leng Max Lehn* Robert Rehner† Alejandro Buchmann
Databases and Distributed Systems
TU Darmstadt, Germany
{cleng,mlehn,rehner,buchmann}@dvs.tu-darmstadt.de

ABSTRACT

Different evaluation methods for distributed systems like prototyping, simulation and emulation have different trade-offs. We present a testbed for Internet applications that supports real-network prototypes and multiple simulators with unchanged application code. To ensure maximum portability between runtimes, a compact but flexible system interface is defined.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications; I.6.8 [**Simulation and Modeling**]: Discrete Event

General Terms

Design, Experimentation, Performance

1. MOTIVATION

Evaluation of prototypes for distributed systems is challenging. A few hundred nodes can be deployed on the Internet with PlanetLab [9], but it is not easy to get repeatable results when using a shared and public infrastructure like the Internet. Emulation testbeds like Emulab [4] are more controllable, but are expensive to acquire and maintain. Packet simulators like ns-3 [3] can run hundreds of nodes on commodity hardware and still give a good approximation of real networks, but become slow if scaled up to larger networks.

In research areas that deal with extremely large networks like peer-to-peer systems or massively multiplayer online games (MMOG) overlay simulators like PlanetSim [2] or ProtoPeer [1] have become very popular. By providing high-level APIs and a rather abstract network model they scale to many thousands of nodes. The downside is that the abstraction leads to reduced realism and the specialization makes it hard or impossible to implement anything beyond peer-to-peer systems. Additionally, most overlay simulators are implemented in Java, rendering them practically useless for applications written in native code (e.g. typical computer games).

*Supported by the DFG Research Group 733, Quality in Peer-to-Peer Systems (QuaP2P).

†Supported by the DFG Research Training Group 1343, Topology of Technology.

SIGCOMM'11, August 15–19, 2011, Toronto, Ontario, Canada.
ACM 978-1-4503-0797-0/11/08.

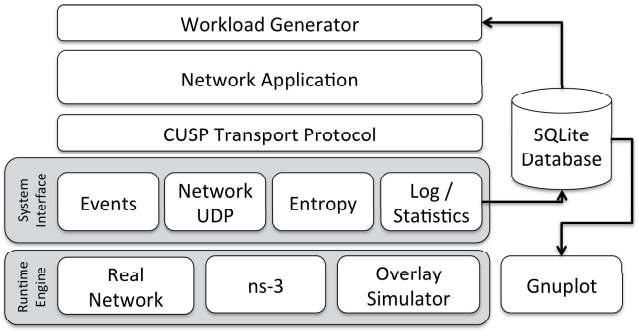

Figure 1: Testbed overview

It seems desirable to combine the advantages of several of the described approaches. On the other hand maintaining several independent implementations of a research prototype would be wasteful. ProtoPeer can be used to generate stand-alone applications for real networks, but only provides an overlay simulator. In our approach we define a system interface that can be used for real networks, low-level, and high-level simulators.

The identification of a common interface that works for all of these scenarios is a useful insight by itself, but our custom high-level simulator provides useful techniques for debugging complex large-scale systems. The system (Figure 1) consists of multiple runtimes which implement the system interface and applications which build on the system interface and (optionally) the CUSP transport protocol [11]. A workload module is used to generate application load. In a simulation the workload and scenario definitions are read from a central database and simulator output is written to the same database.

We have already used the testbed to implement CUSP demo applications [11], BubbleStorm [10], pSense [8], Kademlia [7], and the Planet PI4 online game [6]. The system is implemented in StandardML and provides language bindings to Java and C/C++.

In order to use the testbed environment an application developer must follow a few simple rules. Firstly, the system interface must be used exclusively for the offered functionality and most not be circumvented (e.g. by using the native network socket interface). This ensures that all relevant calls can be redirected to the current runtime engine. Secondly, the application must be written in an event-driven, asynchronous fashion. This enables the simulator engines to run many nodes in parallel without timing conflicts. Al-

though such restrictions have to be considered when porting an existing application to the testbed, porting the non-trivial Planet PI4 game was surprisingly easy.

2. SYSTEM INTERFACE

The system interface consists of four modules: scheduling of events, networking, entropy, and output. An application using the interface must be written in an event-driven fashion in order to be executed by the simulator.

Event scheduling is responsible for scheduling new events and executing them on time. The interface also supports UNIX signals which can be used to trigger application handlers for workload or simulator events (e.g. node shutdown) without breaking the abstraction.

The networking interface is an asynchronous UDP networking interface. It also includes an opaque network address structure. This allows for a relatively smooth transition path from IPv4 to IPv6 addresses.

The entropy interface provides random numbers to the application for cryptographic or stochastic operations. In the real-network it is implemented using operation system means such as /dev/random. In a simulation it provides pseudo random numbers from the simulator's random number generator and thus ensures repeatability of the experiments.

The output interface is used for logging and statistics. The log interface can write short messages to a logging facility. The statistics interface can monitor performance metrics and system parameters measured by the application. In a simulation the output data is written to the scenario database whereas in the real network log files or standard output can be used.

Not strictly a part of the system interface, but being a module shared between many applications, is the CUSP transport protocol. It provides reliable in-order message streams for applications that require more than UDP. Moving the transport protocol implementation out of the runtime has made the system interface much easier to port, since it reduces not only the interface complexity but also the implementation complexity of the runtime (i.e. simulator) itself.

This collection of interfaces has proven to be sufficient for the networking applications we have worked with so far. Nonetheless we consider adding a data storage interface to improve the support for applications that store persistent data between sessions.

3. RUNTIME ENGINES

We currently support three runtime engines: real network, ns-3 [3], and our own overlay simulator. The real-network mode connects the application more or less directly with the operating system. With ns-3 we support a sophisticated and widespread packet-level simulator which can be used to validate results from our custom runtime engines. In day-to-day use we prefer our lightweight overlay simulator which is not as precise but more scalable and easier to configure.

4. OVERLAY SIMULATOR

Our custom simulator is completely event-based and builds on a lightweight end-to-end delay model [5]. It receives the node configuration, session times, and simulation workload from an SQLite database. The output of nodes is written to the same database. This ensures that simulation results are always kept together with the scenario setup that produced the results. Gnuplot scripts can generate statistics plots directly from the database. The log table in the database can be used to track bugs not only between nodes but also forward and backward in time, a feature that step debuggers do not provide. This is especially useful if the conditions that lead to a bug have built up through time.

5. CONCLUSION

The testbed has proven to be a very useful tool for distributed system development and is in day-to-day use in our lab. Being constrained to an interface that also supports the real-network runtime helps to avoid unrealistic shortcuts in the application code that might have happened in a less strict simulation environment. The post-mortem debugging with the log database makes bug-hunting in distributed systems much easier.

6. REFERENCES

[1] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer. ProtoPeer: a P2P toolkit bridging the gap between simulation and live deployement. In *Procs. of Simutools*, 2009.

[2] P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo. Planetsim: A new overlay network simulation framework. In *Software Engineering and Middleware*, LNCS. Springer, 2005.

[3] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *Procs. of WNS2*, 2006.

[4] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the Emulab network testbed. In *Procs. of USENIX*, 2008.

[5] S. Kaune, K. Pussep, A. Kovacevic, C. Leng, G. Tyson, and R. Steinmetz. Modelling the Internet Delay Space Based on Geographic Locations. In *Procs. of PDP*, 2009.

[6] M. Lehn, T. Triebel, C. Leng, A. Buchmann, and W. Effelsberg. Performance Evaluation of Peer-to-Peer Gaming Overlays. In *Procs. of P2P*, 2010.

[7] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Procs. of IPTPS*, 2001.

[8] A. Schmieg, M. Stieler, S. Jeckel, P. Kabus, B. Kemme, and A. Buchmann. pSense - Maintaining a Dynamic Localized Peer-to-Peer Structure for Position Based Multicast in Games. In *Procs. of P2P*, 2008.

[9] N. Spring, L. Peterson, A. Bavier, and V. S. Pai. Using PlanetLab for network research: Myths, realities, and best practices. *Operating Systems Review*, 40(1), 2006.

[10] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. In *Procs. of SIGCOMM*, 2007.

[11] W. W. Terpstra, C. Leng, M. Lehn, and A. P. Buchmann. Channel-based Unidirectional Stream Protocol (CUSP). In *Procs. of INFOCOM*, 2010.