PeerSim: A Scalable P2P Simulator*

Alberto Montresor
University of Trento, Italy
alberto.montresor@unitn.it

Márk Jelasity
University of Szeged and HAS, Hungary
jelasity@inf.u-szeged.hu

1. Introduction

The key features of peer-to-peer (P2P) systems are scalability and dynamism. The evaluation of a P2P protocol in realistic environments is very expensive and difficult to reproduce, so simulation is crucial in P2P research.

PEERSIM is an extremely scalable simulation environment that supports dynamic scenarios such as churn and other failure models. Protocols need to be specifically implemented for the PEERSIM Java API, but with a reasonable effort they can be evolved into a real implementation. Testing in specified parameter-spaces is supported as well.

PEERSIM started out as a tool for our own research. After releasing it under the LGPL open source license, we were pleased to discover that our pragmatic design choices appealed to many other independent research groups: at the time of writing, PEERSIM has been downloaded over 12,000 times and has been used in more than 150 scientific papers, of which only a small fraction has been written by the PEERSIM authors.

It is often said that "eating your own dog food" can be the key to success in an open source project: our own research in diverse P2P application areas ranging from aggregation to topology maintenance, and from synchronization to global optimization has been carried out using PEERSIM [1, 2, 5, 6].

2. Modularity and Configuration

PEERSIM was designed with modularity and ease of configuration in mind. The *network* is modeled as a list of *nodes*; a node has a list of *protocols*, and the simulation has *initializers* and *controls*.

Initializers are executed before the simulation, while controls are executed during the simulation. They may modify or monitor every component. For example, they may add new nodes or destroy existing ones; or they may act at the level of protocols providing them with external input or modifying their parameters. Controls can also be used to passively monitor the simulation; for example, they can report the variance reduction rate during the execution of a diffusion-based aggregation protocol [4], or they may report graph-theoretical properties of overlay topologies, such as diameter, clustering, and so on.

```
network.size 10000

simulation.cycles 100

D 20

protocol.news Newscast
protocol.news.cache D

init.rand WireKOut
init.rand.protocol news
init.rand.k D

control.conn Clustering
control.conn.protocol news
```

Figure 1. A simple PEERSIM configuration file

The simulation *engine* can be cycle-based (protocols are executed is some specified order) or event-based. These components can be fully configured and customized. PEERSIM provides simple components with basic functionalities, but users are allowed to replace them with their own alternative implementations based on their preferences.

Each simulation is specified by a plain text configuration file similar to a Java property file. Properties define implementations (Java classes) of components, and they also specify numeric or string parameters for these components. Configuration files fully specify an experiment, so they can be bundled with simulation code for reproducibility.

PEERSIM has been implemented in Java. This allows us to build experiments at run-time via reading configuration files and dynamically loading classes using Java reflection.

The example configuration file in Figure 1 defines a network composed of 10,000 nodes. Figure 2 illustrates the resulting peersim components. The simulation is run using the cycle-based engine for 100 cycles. Each node runs a protocol called `news` that is implemented by class

*Available from http://peersim.sourceforge.net. In Proc. IEEE P2P 2009, pp 99–100, doi:10.1109/P2P.2009.5284506 . A. Montresor was supported by the European Commission through the NAPA-WINE Project (Grant No. 214412). M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences.

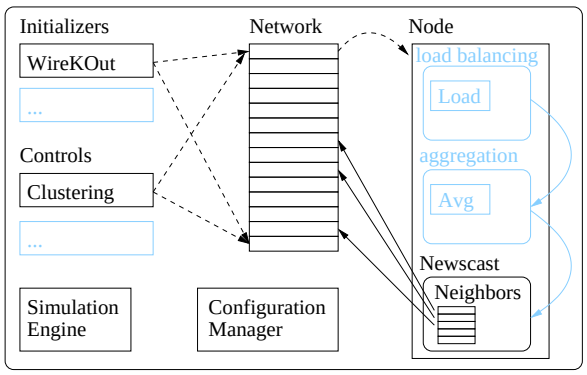

Figure 2. PEERSIM components. Additional protocol layers and components can easily be added (shown in light color).

`Newscast` [6]. The only parameter of `Newscast` is the cache size `cache`, which gets the value of `D` (i.e., 20). The same constant is used in the initializer `rand`, implemented by class `WireKOut` that initializes the overlay links managed by `news` to create a random overlay topology with constant out-degree `k=D`. Finally, observer `Clustering` periodically reports the clustering coefficient of the overlay network managed by `news`.

PEERSIM reads the configuration file and loads the specified classes at runtime. Based on the configuration file, either the cycle-driven or the event-driven simulation engines are loaded. The former, to allow for scalability, uses some simplifying assumptions such as ignoring the details of the transport layer in the communication protocol stack. The latter is less efficient but more realistic. Among other things, it supports transport layer simulation as well. Cycle-based protocols can also be run by the event-based engine, but not vice versa.

3. Feature Highlights

Scalability. Internal data structures have a very small memory footprint. To give a few anecdotic examples: with simple protocols and the cycle-based engine, the limit of network size is practically the entire available memory; networks of more than 10^7 nodes have been simulated in 4G memory. In the other extreme case, the most complex protocols we have simulated using the event-based engine still scale up to 10^5 nodes or more.

Modularity. All components of the simulated system, as well as the observer and initializer components can be freely configured. In addition, the implementation of the components of the simulator itself can be customized. It is possible to replace key components such as the network node and the event queue of the event-based engine, simply by implementing the appropriate interface and adding a line to the configuration file.

Graph abstraction. One strong feature of PEERSIM is its graph abstraction. It can treat overlay networks as graphs and can provide various initializers (random and small-world models, etc.), as well as observers including network diameter, clustering, and connectivity. Overlay network graphs can be exported in various popular formats for drawing and analysis.

Vector abstraction. Existing PEERSIM modules allow developers to enrich their simulations by simply writing a few lines of text in the configuration file. For instance, the `vector` package allows a set of protocol instances located at each of the nodes to be treated as a vector. Vector operations such as calculating the angle between two vectors, or initializing vectors, are supported.

Transport layer and churn. To augment the realism of simulations, PEERSIM can be configured to use trace-based datasets. This feature is supported only in the event-based engine. The transport layer is modeled via a special protocol that provides a message sending service. For example, the King dataset is supported, which models the latency among a collection of geographically distributed nodes [3]. Churn models are available as well, for the cycle-based as well as the event-based model.

References

[1] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor. Firefly-inspired heartbeat synchronization in overlay networks. In *First IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pages 77–86, Boston (MA), USA, 2007. IEEE Computer Society.

[2] M. Biazzini, B. Bánhelyi, A. Montresor, and M. Jelasity. Distributed hyper-heuristics for real parameter optimization. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*, 2009.

[3] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Internet Measurement Workshop (SIGCOMM IMW)*, 2002.

[4] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, August 2005.

[5] M. Jelasity, A. Montresor, and O. Babaoglu. T-Man: Gossip-based fast overlay topology construction. *Computer Networks*, 2009. in press, DOI: 10.1016/j.comnet.2009.03.013.

[6] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25(3):8, August 2007.