

ProtoPeer: from Simulation to Live Deployment in One Step

Wojciech Galuba, Karl Aberer

Ecole Polytechnique Fédérale de Lausanne (EPFL)

Lausanne, Switzerland

{wojciech.galuba,karl.aberer}@epfl.ch

Zoran Despotovic, Wolfgang Kellerer

DoCoMo Euro-Labs

Munich, Germany

{despotovic,kellerer}@docomolab-euro.com

1 Introduction

Simulators are a commonly used tool in peer-to-peer systems research. However, they may not be able to capture all the details of a system operating in a live network deployment. Transitioning from simulation to the actual system implementation is a non-trivial and time-consuming task, a problem that we propose to solve in this paper.

The paper’s secondary motivation stems from the recent observations made by Naicken et. al [1] who examined 287 papers on P2P systems and concluded that there is almost no simulator code reuse and little standardization of the common practices. We would like to contribute our toolkit and share the design expertise with the P2P community.

We present ProtoPeer¹, a peer-to-peer systems prototyping toolkit that allows for switching between the event-driven simulation and live network deployment without changing any of the application code. ProtoPeer exports a set of APIs for message passing, message queuing, timer operations as well as overlay routing and managing the overlay neighbors. Users can plug in their own custom implementations of most of the parts of ProtoPeer including custom network models for simulation and custom message passing over transports other than the default TCP/UDP.

Applications implemented using ProtoPeer are divided into modules each encapsulating a separate piece of the message passing functionality. The modules can be reused and composed to achieve the desired system behavior. ProtoPeer has a unified system-wide infrastructure for measurement logging and aggregation, event injection and managing evaluation scenarios. The simulator scales to tens of thousands of peers and gives accurate predictions closely matching the live network measurements.

2 Architecture

Message passing & timers. In ProtoPeer the system is composed of a set of peers that communicate with one another by passing messages. Each application defines its set

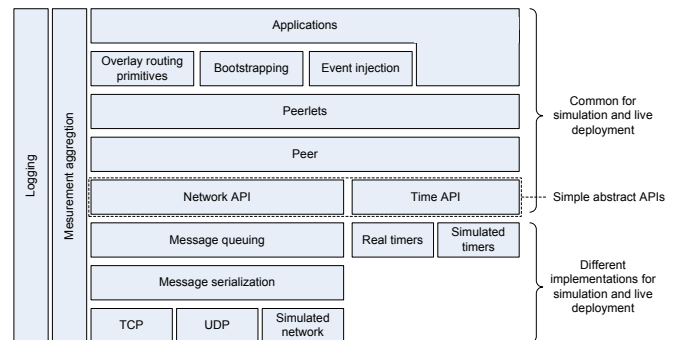


Figure 1. The ProtoPeer architecture. The time API and the networking API consist of only a few basic abstractions and methods and form the narrow "waist" of the ProtoPeer's hourglass architecture. The upper part of the hourglass are all the components that use the time and networking APIs. The peer provides the runtime context for the peerlets in which the various parts of the peer's functionality are encapsulated. The lower part of the hourglass are the concrete implementations of the abstract time and networking APIs. Switching from the simulated system to the actual system is as simple as switching from one time & networking implementation to another.

of messages and message handlers. An application typically also defines a set of timers and handlers for the timer expiration events. All the application logic in ProtoPeer is called from within the timer and message handlers.

Networking & time abstraction. One of the main goals of ProtoPeer is to be able to switch between simulation and live network deployment without changing any of the application's code. The key architectural feature that enables this are the abstract time and networking APIs (Fig. 1). The APIs allow for only a small number of basic operations: creation of timers for execution scheduling and creation of network interfaces for sending and receiving messages. When switching from the simulated run to the live run the simulated time and networking implementations are simply swapped with the implementations using real timers and TCP or UDP networking.

Event-driven execution. Execution in ProtoPeer pro-

¹<http://protopeer.epfl.ch>

gresses by calling event handlers in response to time and networking events. For example, when a timer expires an appropriate handler is called for that event. The handler might send a message or schedule other timers, which subsequently trigger other handler calls and so on. The same execution model is used both during the simulation and the live run.

Network modelling. During simulation the network needs to subject the messages to realistic delay and loss. Loss and delay modelling are encapsulated in the `NetworkModel` interface in `ProtoPeer`. Users can provide their own implementations of that interface. There are several implementations already available, including simple uniformly distributed delay model, the Euclidean model (i.e. delay between nodes proportional to their distance in the Euclidean space) or the delay matrix model into which arbitrary delay matrices can be loaded (e.g. based on the King dataset²).

Overlay modelling. Research on peer-to-peer systems has produced many ways of constructing and maintaining overlays. To support the wide range of overlays, `ProtoPeer` defines an abstract `PeerIdentifier` that is used throughout the system. The different overlay implementations override it with concrete implementations of their ID space and the associated distance metric. Basic message routing primitives are available including recursive and iterative routing as well as different forms of constrained flooding (e.g. TTL-limited).

Peerlets. A peer in the peer-to-peer system typically implements more than one piece of the message passing functionality, for example, the bootstrap protocol, overlay maintenance or DHT key replication. In `ProtoPeer` the message passing logic and state of each of the protocols is encapsulated in components called *peerlets*. Peers are constructed by putting several peerlets together. The peerlets can also be removed or added at runtime. The peerlet-based approach has all the advantages of any other modular design: (i) encapsulation and isolation of message passing functionality, (ii) functional composability, (iii) reusability and (iv) ease of unit testing.

3 Tools

Measurement infrastructure. Obtaining reliable and accurate measurements is an important, if not the most important part of any peer-to-peer system evaluation. Measurements need to be instrumented in the application code, logged, aggregated from all the peers in the system, analyzed and optionally plotted. `ProtoPeer`'s measurement infrastructure covers all of these tasks. The measurements are instrumented by making calls to the measurement API in the appropriate places in the application code. The system

computes the basic statistics on-the-fly: average, sum, variance etc. The statistics can be computed at various aggregation levels: per peer, per time window and per measurement "tag" and plotted using our visualization tool.

Event injection & scenarios While evaluating the peer-to-peer system there is frequently a need to test the system's response to various, often exogenous events, e.g. peer arrivals and departures (i.e. churn), user actions or, more commonly, failures. `ProtoPeer` provides a simple but general mechanism for event injection. The events are specified in triples consisting of time, the set of unique peer indices to be affected and the method to call. Despite its simplicity this way of describing events is expressive enough to cover most of the common use cases. Events are grouped into scenario files which improve experiment repeatability and can easily be sliced and diced with the common text processing tools.

4 Implementation

`ProtoPeer` is developed in Java, which has been chosen for its popularity, ease-of-use and availability of libraries. Networking is implemented using Apache MINA³, a high-performance thread-pooled asynchronous networking framework. Messages can be sent either over UDP or TCP. The choice which of the two to use can be made by the application on the per-message basis.

5 Demonstration

In our demonstration we will use a bidirectional Chord implementation running on 350 peers. We will run the same overlay routing workload both in simulation and in live PlanetLab deployment over several minutes. To showcase `ProtoPeer`'s event injection capability we will simulate churn and inject a large-scale routing failure to test the system's response.

In both settings, the simulation and the live deployment, measurements will be aggregated and fed into our measurement plotting application. We will demonstrate that the results from the `ProtoPeer` simulator closely match those from the live deployment. In addition, the statistics gathering capabilities of the measurement infrastructure and the automated deployment tools will be demonstrated.

References

- [1] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *SIGCOMM Comput. Commun. Rev.*, 37(2):95–98, 2007.

²<http://pdos.csail.mit.edu/p2psim/kingdata/>

³<http://mina.apache.org/>