ProtoPeer: Distributed Systems Prototyping Toolkit

`http://protopeer.net`

Wojciech Galuba, Karl Aberer
Ecole Polytechnique Fédérale de Lausanne (EPFL)
`firstname.lastname@epfl.ch`

Zoran Despotovic, Wolfgang Kellerer
DOCOMO Euro-Labs, Munich, Germany
`lastname@docomolab-euro.com`

I. Introduction

Simulators are an excellent tool for quick systems prototyping and large-scale parameter sweeps, but they cannot match the accuracy of live network evaluations. Switching from the simulation to the actual implementation often requires a considerable development effort. Network I/O implementation, measurement instrumentation, deployment automation and distributed debugging are among the most time consuming tasks that the developers face. Moreover, large parts of the application code existing in the simulator are not reused in the actual system. Bridging the gap between the simulation and the actual system deployment was the primary motivation for developing ***ProtoPeer*** [1], the framework presented in this talk.

In ProtoPeer, the developer can switch between the simulation of a P2P system to its deployment on the actual network without changing a single line of code. This dramatically speeds up the implement-evaluate-reimplement cycle and allows for rapid system prototyping. Most of the major bugs and performance problems are caught early on during the simulation while the more time-consuming live deployment is used for the accurate evaluation of the final system.

II. Overview

Message passing & timers. ProtoPeer is event-driven and the system is composed of a set of peers that communicate with one another by passing messages. Each application defines its set of messages and message handlers. An application typically also defines a set of timers and handlers for the timer expiration events. Most of 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 code. The key architectural feature that enables this are the abstract time and networking APIs[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. These simple APIs serve as the key building block for the rest of the ProtoPeer and form the "waist" of the framework's hourglass architecture (Fig. 1).

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. Users can also provide alternative time and networking implementations, e.g. other transports or layer on top of other simulation frameworks. We have, for example, successfully integrated ProtoPeer on top of JiST/SWANS[2], a MANET simulator.

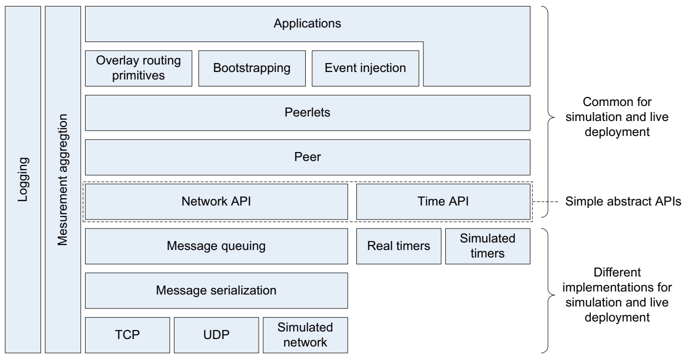

Fig. 1. **The ProtoPeer architecture.** The simplified time and networking APIs form the waist of the ProtoPeer's architecture. The application complexity grows from these APIs up, while the simulation and live networking complexity grows from the APIs down.

Network modeling. During simulation, the network model needs to subject the messages to realistic delay and loss. Loss and delay modeling are encapsulated in the `NetworkModel` interface in ProtoPeer. Users can provide their own implementations of that interface. There are several implementations already available, including the 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 dataset[3]).

ProtoPeer also has a MaxMin flow-based network model, which takes into account the message and takes care of bandwidth allocation in the network. This model is especially useful for simulating bandwidth-bound applications such as BitTorrent, while the other simpler and faster models mentioned above can be used for delay-bound applications such as DHTs.

Peerlets. A peer in a distributed system typically implements more than one piece of message passing functionality.

[1]Tutorial at: `http://protopeer.epfl.ch/wiki/IntroTutorial`

[2]`http://jist.ece.cornell.edu/`

[3]`http://pdos.csail.mit.edu/p2psim/kingdata/`

```
#set up the churn sequence
4        14.3     Peer.start()
3        15.1     Peer.stop()
1        36.9     Peer.start()
4        44.4     Peer.stop()

#inject a failure at 150s on 10 peers
0-9      150.0    Peer.Router.setDropMessages(true)
```

Fig. 2. **An example scenario file.** Each scenario file consists of three columns. The first one specifies the peer index (or a range of indices) that uniquely identify the affected peer. The second column is the number of seconds since the beginning of the simulation when the event should be injected. The last column indicates the method to be called. Churn can be simply defined as a sequence of calls to the peer's start and stop methods. Calls can be made to any method of the peer or its peerlets. Users can define their own methods and call them in the scenario files, which makes event injection a multipurpose tool.

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 peerlets, just as the applets or servlets, have the familiar init-start-stop lifecycle. The peer provides the execution context for all of the peerlet instances it contains. The peerlets can discover one another within that context and use one another's functionality.

The peerlet-based approach has all the advantages of any other modular design. Firstly, the message passing functionality is conveniently encapsulated in building blocks with well defined behavior. The blocks can be composed to achieve the desired peer functionality. Certain functionality can be easily enabled or disabled depending on the context (e.g. debug mode vs. evaluation mode). Secondly, peerlets can be reused across applications. Peerlets can export well defined interfaces e.g. a DHT interface, which can have several implementations that can be easily swapped one for another. Lastly, peerlets can be unit tested either in isolation or with other peerlets as mock objects.

III. System evaluator's toolkit

Event injection & scenarios. While evaluating a distributed 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 1) time, 2) the set of unique peer indices to be affected and 3) the method to call. Despite its simplicity, this way of describing events is expressive enough to cover most of the common use cases.

A set of events defines a *scenario*. The scenario files can be generated, merged and filtered in various ways using the common text processing utilities. Scenarios are an important tool for systematizing the evaluation process and ensuring high experiment repeatability. Like everything else in ProtoPeer, the scenarios work in the same way both in simulation and during live deployment, no changes to the code or the scenario files are necessary when switching between the two.

Measurement infrastructure. Obtaining reliable and accurate measurements is an important, if not the most important part of any peer-to-peer system evaluation. The application code has to be appropriately instrumented, the measurements need to be logged, aggregated from all the peers and analyzed. ProtoPeer's measurement infrastructure helps with all of these tasks. Instrumentation is done by doing calls to the measurement API in the appropriate places in the application code. Just as in the case of other logging frameworks such as log4j, the measurement gathering can be turned off reducing the measurement API calls to practically none.

Once the measurement code is added to the application code it does not have to be changed when switching between the simulation and live deployment. During the simulation, the measurements are accumulated in the system-wide root logger. During live deployment the measurements are dumped to a file on each node locally and either during or after the experiment the files can be merged into a single log, which can be queried for measurement aggregates.

While the measurements are accumulating, the system computes the basic statistics on-the-fly: average, sum, variance etc. This allows for extremely compact representation of the measurements. Optionally all logged values can be kept, which later on permits the computation of other statistics such as percentiles. The statistics can be computed at various aggregation levels: per peer, per time window and per measurement tag. Measurement ***tags*** are objects used to "mark" the values that are reported to the measurement logger. The user can request an aggregate performed over all the measurements that match a certain tag or a set of tags. We found that in practice the simple tagging-based measurement infrastructure covers the vast majority of needs. For more sophisticated measurements, the user can still rely on the usual text-based measurement pipeline using log4j, grep etc.

IV. Implementation

ProtoPeer is developed in Java. Networking is implemented using Apache MINA[4], a high-performance networking framework. Its event-driven design and the use of non-blocking I/O fits well into ProtoPeer. Messages can be sent either over UDP or TCP. During the live run messages are serialized using a custom optimized protocol that is considerably less verbose than the standard Java serialization. The simulation runs in a single JVM and can scale to tens of thousands of peers on a 3GB latptop. Live runs do not use any centralized components once deployed and can in theory be scaled indefinitely.

References

[1] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer. ProtoPeer: A P2P Toolkit Bridging the Gap Between Simulation and Live Deployment. In *2nd International Conference on Simulation Tools and Techniques (SIMUTools'09)*, 2009.

[4]`http://mina.apache.org/`