ProtoPeer: A P2P Toolkit Bridging the Gap Between Simulation and Live Deployment

Wojciech Galuba, Karl Aberer (EPFL)
Zoran Despotovic, Wolfgang Kellerer (Docomo Euro-Labs)
+ many sourceforge contributors
The system evaluation path

An idea → Message passing spec → Analytical models → Simulation → Emulation → Live deployment

© 2008 EPFL, visit: http://protopeer.epfl.ch
Solution

Write application once and switch between:

- Simulation
- Emulation
- Live deployment

© 2008 EPFL, visit: http://protopeer.epfl.ch
How does it work?

- Simulate under different network models
- Fast sys eval cycle, easy debugging
How does it work?

- No changes in the code

Run live or emulate on a real network stack (TCP, UDP, ...)
- Accurate performance eval in the wild

© 2008 EPFL, visit: http://protopeer.epfl.ch
How does it work?

- No changes in the code

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking API</td>
</tr>
<tr>
<td>Your own networking and time implementation</td>
</tr>
</tbody>
</table>

- Run P2P apps on other networks
- Interface with other simulators
- Easy because Net and Time APIs are narrow
Networking API

- Communication: message passing
- Messages are Java classes:

  ```java
  public class PutRequest extends Message {
      int requestID;
      Object key;
      Object value;
  }
  ```

- Sending:

  ```java
  sendMessage(destAddr, new PutRequest("key", "value"));
  ```

- Message serialization and queuing taken care of
Networking API

- Networking is asynchronous
  - All calls non-blocking
  - Callbacks on: send completion, exceptions, message receive

```java
public void handleIncomingMessage(Message message) {
    if (message instanceof PutRequest) {
        PutRequest request = (PutRequest) message;
        //store the key-value pair in some local storage
        localStorage.put(request.key, request.value);
        //send the ack back to the source
        sendMessage(request.getSourceAddress(),
                    new PutResponse(request.requestID));
    }
}
```
Networking extensibility

- Can plug in new networking implementations
  - Easy to do, networking API is simple
  - Can run ProtoPeer apps on top of JiST/SWANS (a MANET simulator)

- Can plug in new network models
  - To model loss and delay during simulation
  - For delay-sensitive apps:
    - King latency model
    - Loss and delay snapshots from PlanetLab
    - DIMES coming soon…
  - For bandwidth-sensitive apps
    - Flow-based model
    - Max-min bandwidth allocator

© 2008 EPFL, visit: http://protopeer.epfl.ch
Time API

- Create timers and schedule them
- Callbacks on timer expiry
- During simulation uses scheduled events
- During live runs uses Java timers
Peer and peerlets

- The peer provides the execution context for the peerlets
  - Clock, network interface
- Peerlets are pieces of state and message passing functionality
  - Reusability
- \texttt{init()}-\texttt{start()}-\texttt{stop()} lifecycle
- Peerlets can discover one another via the peer
  - Composability

© 2008 EPFL, visit: http://protopeer.epfl.ch
Peerlet examples

- **BootstrapClient**
  - contacts the BootstrapServer on `start()` and obtains the fingers of initial overlay neighbors

- **BootstrapServer**
  - waits for enough peers to join the network and then wires them up in some topology

- **CrashStopFailureDetector**
  - periodically sends heartbeats to overlay neighbors
  - responds to heartbeats from the neighbors
  - removes dead neighbors
Measurement API

- Reliable, repeatable measurements are important
  - Though measurements typically instrumented in an ad-hoc way
- In ProtoPeer: automated network-wide data aggregation
- Basic aggregates computed:
  - sum, min, max, average, median, percentile
  - stored in a single queryable file
- Different aggregation levels:
  - Per peer, per measurement tag, per measurement epoch
- Advantages:
  - Once added, the same measurement code is used both during the simulation and during live deployment
  - Also a powerful debugging tool

© 2008 EPFL, visit: http://protopeer.epfl.ch
Event injection

- Important aspect of sys eval
  - Often implemented in an ad-hoc way
  - Event injection code interspersed with application code…
- In ProtoPeer: more systematic approach
  - Event injection separated from the evaluated application
- Event spec:
  - `<what peers> <when> <method call>`
Scenarios – sets of events

```bash
#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)
```

- Specified in scenario files
  - Easy scenario management
  - Repeatability of experiments
  - **Works the same in live and sim**

© 2008 EPFL, visit: [http://protopeer.epfl.ch](http://protopeer.epfl.ch)
Implementation details

- Java 1.6
- Event-driven design, everything async
  - Both the simulator and the live run
  - Easy allocation of events to threads
- Apache MINA for async TCP and UDP networking
  - Not a problem to implement over other transports
- Using Java serialization
  - Currently looking at protobuf and thrift as alternatives
- log4j for logging
Performance

- Simulator scales to tens of thousands of peers
  - For practical applications
  - Currently memory-bound, not CPU-bound

- Message pipeline throughput
  - 2k-10k messages per second
  - Thrift and protobuf likely to improve throughput
Basic accuracy test

- **Goal**: validate ProtoPeer’s delay and loss model used during sim
- **First build a delay-loss model:**
  - Measure loss and delay on all links between 350 PlanetLab nodes
  - Build a delay-loss matrix based on measurements
- **Then test case:**
  - Chord, distributed key-value storage, 350 nodes
  - Each node querying for random keys
  - Multi-hop query routing (~4 hops on average)
Accuracy – loss

![Graph showing accuracy loss over time with simulation and PlanetLab data.](http://protopeer.epfl.ch)
Accuracy - delay

- Simulation, 95th percentile
- PlanetLab, 95th percentile
- Simulation, median
- PlanetLab, median

© 2008 EPFL, visit: http://protopeer.epfl.ch
Compared to others – the bad

- **Small suite of implemented well-known protocols**
  - Only Chord and BitTorrent for now… awaiting your contributions :)
  - Long term vision: a library of peerlets, lego blocks

- **Slightly more verbose protocol representation**
  - Java vs. OverLog (P2) vs. Lua (SPLAY)
  - However, no need to learn a new language

- **No packet-level simulation detail**
  - Only message-level, sufficient accuracy and fast
  - Can potentially integrate with e.g. ns2

- **No CPU or I/O abstraction**
  - Only time and networking
Compared to others – the good

- Designed from ground up for switching between sim and live
  - Speeds up the development & sys eval cycle
  - Actual running systems at the end of prototyping
- Time and Networking abstraction
  - Easy to add new implementations
  - Network models as plug-ins
- Event injection and measurement
  - Systematizes the evaluation process
  - Instrumentation does not change between live and sim
- Java
  - Cross-platform, familiar language, wealth of libraries
  - Easy: two undergrads developed a DHT in a week

© 2008 EPFL, visit: http://protopeer.epfl.ch
http://protopeer.net