



PeerSim P2P Simulator: Short Howto and Hands-on Proposed Experiments

Gian Paolo Jesi (jesi@cs.unibo.it)

First Summer School on Aspects of Complexity

July 18th, 2005 Bertinoro





Summary:

- PeerSim simulator introduction
- PeerSim basic components
- PeerSim configuration model
- Suggested experiments





PeerSim Introduction (1/4)

PeerSim is an open source, Java based, P2P simulation framework aimed to develop and test any kind of P2P algorithm in a dynamic environment.

<http://peersim.sf.net>

Main features:

- Scales up to 1 million peers;
- Highly configurable;
- Open architecture and component based;





PeerSim Introduction (2/4)

- Documentation: JavaDoc and tutorials available online.
- Support: help web based mailing list available on sourceforge.
- Peersim popularity is growing day by day: houndreds of dowloads per months.
- All the algorithms developed in the Bison project have been developed and tested on Peersim.





PeerSim Introduction (3/4)

Peersim supports 2 kind of simulation:

- Cycle based: the simulation runs in a sequential order and, in each cycle, each protocol can run its behaviour.
- Event based: support for concurrency; a set of events (messages) are scheduled and the node protocols are run accordingly to the message delivery.





Peersim Introduction (4/4)

Peersim limits:

- Ignores the details of the transport communication protocol stack.
- Messages are not modeled by default (we invoke object methods instead), but the developer is free to implement them.





Basic Entities

- Node: basic element of a network, is an interface defining basic operations on neighbor views. A basic node impl. is given.
- Network: global array of Nodes, it represents the whole participants set.
- Scheduler: objects that wrap any Protocol, Dynamics and Observer to supply scheduling facilities during the simulation (e.g.: Initializers are automagically scheduled at the beginning).





Components

- Peersim is component based (nothing to deal with CORBA or .NET component idea).
- Components are “pluggable”.

Main Component types:

- Protocol
- Dynamics (and Initializers)
- Observer





Protocol Component

- Holds the core algorithms that a designer have to write.
- Means implementing the a very simple Java interface (only 1 method).
- Many protocols (no upper bound) can run on every node.
- Each protocol has an identifier (ID) (NOTE: this may cause troubles to newbies).
- A protocol is supposed to use only *local* informations.





Dynamics Component

- Used to add dynamism; they have global view of the system.
- Means implementing the `peersim.dynamics.Dynamics` interface (only 1 method).
- A Dynamics can be used as an Initializer (e.g.: to initialize peer views).
- No limits on the number of dynamics in a simulation.
- Examples: used to reset node status or to simulate random node crashes.





Observer Components

- Used to log important informations about the system; they have global view.
 - An observer can use the peersim internal log facility to produce consistent data log during simulations.
- Means implementing the `peersim.reports.Observer` interface (only 1 method).
- No limits on the number of observers in a simulation.





Configuration

A Peersim configuration file is a **standard text file** and can manage the following:

- Any peersim component can be defined and configured.
- Any component can be fine scheduled.
- The network size, the length of the simulation (in cycles).
- The Node class type can be redefined.
- Simulation details (e.g., shuffling, seed,...).
- Number of distinct experiments.





Configuration Example

■ Example:

```
# random.seed 1234567890  
simulation.cycles 30  
simulation.shuffle  
overlay.size 50000  
overlay.maxsize 100000
```

```
protocol.0 peersim.core.IdleProtocol  
protocol.0.degree 20
```

```
protocol.1 example.aggregation.AverageFunction  
protocol.1.linkable 0
```

```
init.0 peersim.init.WireRegularRandom  
init.0.protocol 0 17  
init.0.degree 20
```

```
init.1 example.loadbalance.LinearDistributionInitializer  
init.1.protocol 1  
init.1.max 100  
init.1.min 1
```

```
observer.0 example.aggregation.AverageObserver  
observer.0.protocol 1
```





Proposed Experiments

- Experiment 1: aggregation protocol based;
 - File: `configs/config-example1.txt`
- Experiment 2: aggregation + newscast protocol based; added network dynamism;
 - File: `configs/config-example2.txt`
- Experiment 3: newscast (GUI) based;
 - File: `configs/ncast-viz.txt`
- Experiment 4: T-Man (GUI) based;
 - File: `configs/config-tman.txt`
- Experiment 5: load balancing protocol.
 - File: `configs/config-lb.txt`





Experiments Package Setup

- Download the package from:
<http://jesi.web.cs.unibo.it/bertinoro-ca.tar.gz>
- Unpack:
 - > `gunzip bertinoro-ca.tar.gz`
 - > `tar -xvf bertinoro-ca.tar`
- To compile (NOTE: everything is pre-compiled, you should not need this step):
 - > `make clean; make`
- To run an experiment:
 - > `./runpsim configs/filename`





Experiment 1 (1/3)

- Regards the **aggregation** family algorithm.
 - Epidemic-style protocol;
 - Computes a specific function (**avg**, max, min,...) on a numeric value holded at each node.
 - At each cycle, every node perform the same computation with a (randomly) selected neighbour peer: $(a + b)/2$.
 - Aggregation poses no requirements on the topology management (IdleProtocol is used in the experiment).





Experiment 1 (2/3)

■ Output:

```
observer.0 1 7940.027373303268 0.5500957320321409 25000.5 49339.0 381.0 50000 50000
```

```
Simulator: cycle 1 done
```

```
....
```

```
observer.0 10 44.445527015609834 0.0030792456461059543 25000.499999999953 25332.682533285497 24760.754959920294 50000  
50000
```

```
Simulator: cycle 10 done
```

```
observer.0 11 24.96928988636488 0.001729905849511987 25000.499999999767 25148.214169611645 24809.098696150555 50000 50000
```

```
Simulator: cycle 11 done
```

```
observer.0 12 14.03574469892769 9.724151935210168E-4 25000.49999999991 25083.878262220736 24902.32619746952 50000 50000
```

```
Simulator: cycle 12 done
```

```
observer.0 13 7.956071774801109 5.51207309659272E-4 25000.5000000000306 25057.278591436687 24953.120616898042 50000 50000
```

■ Comments:

- in few cycles all nodes have a very good avg estimation.
- the conv.speed is exponential and network size agnostic.





Experiment 1 (3/3)

- Suggested changes to the original configuration:
 - Change the network size:
 - overlay.size [1000:200000]
 - overlay.maxsize 200000
 - Change the node variable initialization distribution:
 - follow the comments in the file.





Experiment 2 (1/4)

- Adds the **Newscast** protocol and **dynamism**.
- Newscast topology management protocol:
 - Epidemic, very robust to failures
 - Each node has partial view of the system modeled as a fixed size (k) set of *descriptors* (IP, timestamp).
 - Each node selects a random node and exchanges with it the view.
 - The current and received views are merged and the k *freshest* descriptors are kept.
 - *Very low diameter* and it is very close to a random graph with degree k .





Experiment 2 (2/4)

- The dynamism can remove or inject nodes at predefined time intervals:

```
dynamics.0 peersim.dynamics.GrowingNetwork
```

```
dynamics.0.add -500
```

```
dynamics.0.minsize 4000
```

```
dynamics.0.from 5
```

```
dynamics.0.until 10
```

```
dynamics.0.init.0 peersim.dynamics.WireRegularRandom
```

```
dynamics.0.init.0.protocol 0
```

```
dynamics.0.init.0.degree 20
```





Experiment 2 (3/4)

■ Output:

...

```
observer.0 5 1.597776119033385 0.05590598778529507 50.500000000000036 58.303603070390565 41.510388381159444 50000 50000  
observer.0 6 0.9232944079564567 0.032305956559591294 50.49936622637051 57.90776052413331 45.306477252157705 49500 49500  
observer.0 7 0.542214548664622 0.018972019655039203 50.49931087611531 53.47750571418749 45.88871715103222 49000 49000  
observer.0 8 0.31808485323669167 0.011129749473597116 50.499478364061574 52.58743918821934 48.61734330090904 48500 48500  
observer.0 9 0.18946906714915338 0.006629499106633283 50.49968576518841 51.57264624450603 49.32535107602454 48000 48000  
observer.0 10 0.11302895281047541 0.003954869008199952 50.499758933914315 51.31609359892657 49.7155485175943 47500 47500  
observer.0 11 0.06756171353151115 0.0023639759578653117 50.49975893391354 50.94119493547392 49.84533838486739 47500 47500  
observer.0 12 0.04072969167393905 0.0014251268485597466 50.49975893391407 50.80368426088714 50.21489282908294 47500 47500
```

■ Comments:

- the conv.speed is still exponential in case of failures;





Experiment 2 (4/4)

- Suggested changes to the original configuration:
 - Change the network size;
 - Use IdleProtocol instead of Newscast (c&p from example1);
 - Change the number of killed nodes [500:2000] per cycle.
 - Change the node variable initialization distribution;
 - Try the GraphStats observer to track the avg path length and clustering coeff;





Experiment 3 (1/3)

- Visualizes in a GUI the newscast generated graph in a step-by-step manner.
- The newscast trend to clustering is visible.
- It is possible to kill individual nodes.

```
observer.0 peergui.GUIObserver
```

```
observer.0.directed
```

```
# TRY DIFFERENT LAYOUTS:
```

```
#observer.0.layout org.jgraph.layout.SpringEmbeddedLayoutAlgorithm
```

```
observer.0.layout org.jgraph.layout.CircleGraphLayout
```

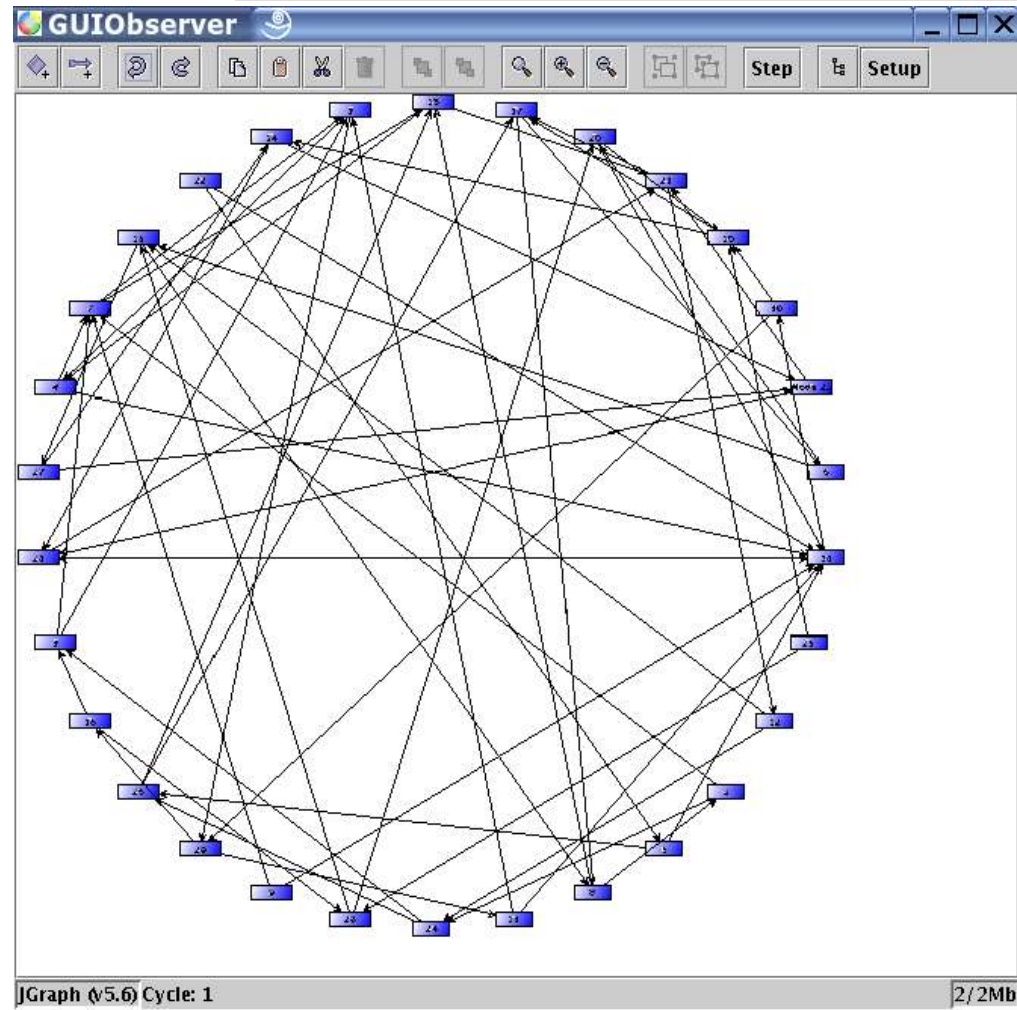
```
observer.0.protocol 0
```

```
observer.0.degree 2
```





Experiment 3 (2/3)





Experiment 3 (3/3)

- Suggested changes to the original configuration:
 - Change the network size;
 - Change the initial overlay initialization (star, scale-free):
 - follow the comments in the file.
 - Change the visualization layout.





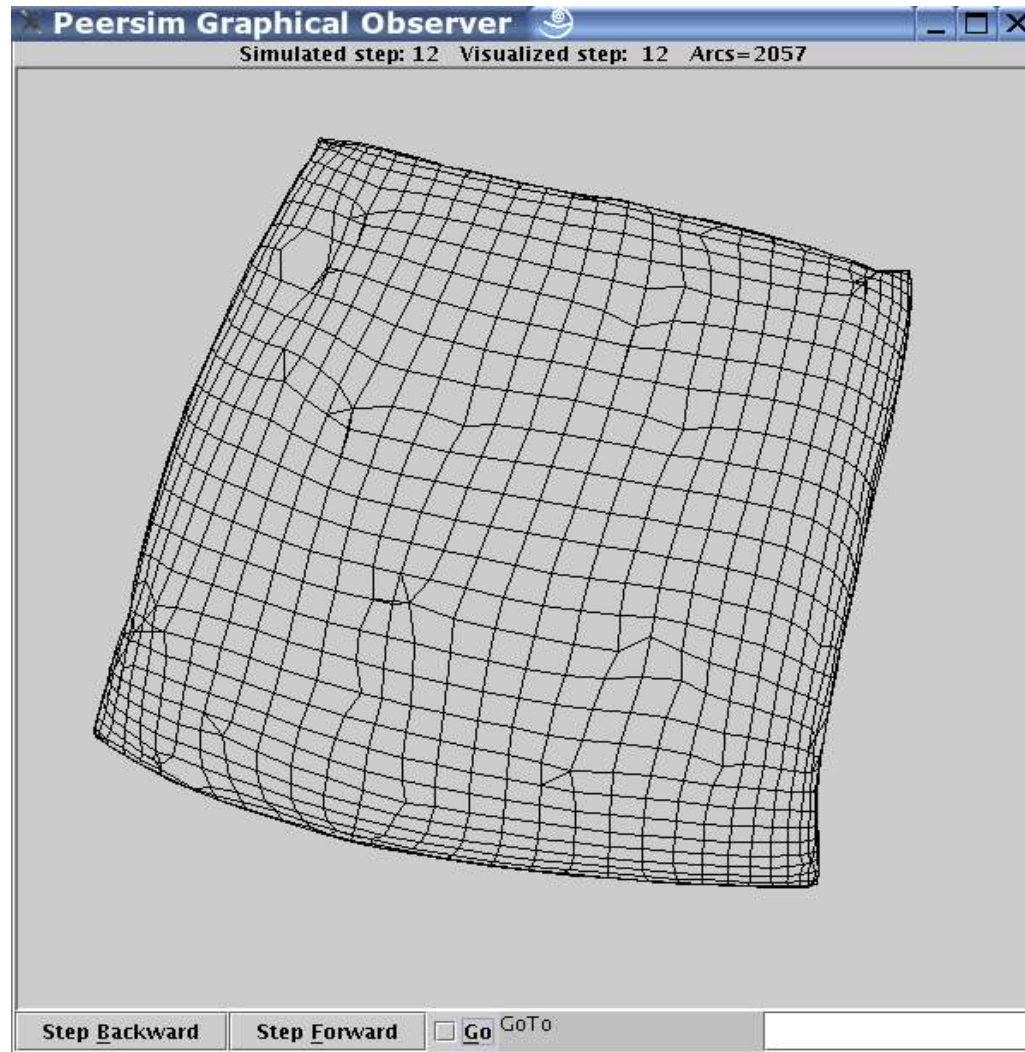
Experiment 4 (1/3)

- Visualizes a **T-Man** protocol overlay emergence.
- T-man:
 - Is an epidemic protocol that can build and maintain distinct classes of topologies.
 - Nodes exchange their view and sort the node *descriptors* according to a *ranking function*.
 - Different ranking functions lead to a different kinds of topology.
- The gui shows the creation of a grid or a line topology.





Experiment 4 (2/3)



18/07/05



Experiment 4 (3/3)

- Suggested changes to the original configuration:
 - Change the network size;
 - Change the topology type:
`protocol.1.dist topman.dist.Linear` or
`protocol.1.dist topman.dist.Grid2d`
 - change the visualization degree:
`protocol.2.observer.0.degree k` with `k`
[2:6].





Experiment 5 (1/3)

- This experiment file aims to compare two simple **load balancing** algorithms and to show that the knowledge of the avg. system load improves the balancing process.
- A *unit* is the base “load” quantity that can be moved at each step.
- The protocol classes are the following (see file comments):
 - `example.loadbalance.AvgBalance`
 - `example.loadbalance.BasicBalance`
- Based on a *push-pull* paradigm.





Experiment 5 (2/3)

- Each node in the system has 2 parameters:
 - *Load*: current load expressed in units, is the value to be balanced with the other peers.
 - *Quota*: allowed load units a node can move at each cycle.
- The AvgBalance protocol is designed to take advantage of a Newscast layer.
- The advantage of the AvgBalance protocol is not in convergence speed, but in *quota consumption*.





Experiment 5 (3/3)

- Suggested changes to the original configuration:
 - Change the network size;
 - Compare BasicBalance + IdleProtocol VS AvgBalance + Newscast;
 - Change the load distribution (linear or peak);
 - Change the overlay topology initialization (star, scale-free).





Your Turn

Good experimentations!



18/07/05



- Bison project publication list:
<http://www.cs.unibo.it/bison/pubs/list.shtml>
- Bertinoro CA experiment package:
<http://jesi.web.cs.unibo.it/bertinoro-ca.tar.gz>
- Peersim site: <http://peersim.sf.net>
- Peersim tutorials:
<http://peersim.sourceforge.net/#docs>
- These slides:
<http://jesi.web.cs.unibo.it/bertinoro-ca.pdf>

