The 3rd Mandatory Programming Assignment:

Gossiping with PeerSim

INF5040 Autumn 2017

Objective

- Development, debugging and simulation of a P2P protocol.
- Using the PeerSim simulator

Scope

PeerSim is a tool for simulating P2P protocols in extremely large scale settings (millions of nodes). Since experimenting with such protocols in real deployments may be very difficult and time consuming, the simulator can provide the means to abstract most of the system complexity and remove the deployment cost. The simulator also makes it easier to prototype a P2P protocol and to evaluate its desired properties with different settings.

In this assignment you are asked to implement a simple **gossiping protocol** in PeerSim. The protocol builds a **P2P overlay** and keeps it connected over time. The protocol is very simple: each peer knows a small, continuously changing set of other peers, called its neighbors, and occasionally contacts a random one to exchange some of their own neighbors. The exchange assures the overlay will stay connected over time and well-mixed (exhibiting important properties of random graphs).

Technical features

The neighbor exchange protocol is called ***Shuffling Protocol***, and was first introduced by Stavrou et al. [1]. The protocol is described bellow:

Each peer in the system maintains a neighbor list in a small cache of c entries (The peers don't need to know about all the other peers in the system). A cache entry contains the address of another peer in the overlay, so that the peers can communicate. Each peer P periodically initiates a neighbor exchange operation, known as *shuffle*, by executing the following six steps:

1. Select a random subset of l neighbors ($1 \leq l \leq c$) from P's cache, and a random peer, Q, within this subset. Where l is a system parameter, called *shuffle length*.
2. Remove Q's entry from the P's cache and from the subset. Add P's address to the subset.
3. Send the updated subset to Q as a shuffle request.
4. Receive from Q a shuffle reply containing a subset of no more than l of Q's neighbors, or a shuffle reject indicating that Q does not want to participate in the shuffle operation. If the request is rejected, just add Q back to the cache.
5. If the request is accepted, discard from the received subset entries pointing to P, and entries that are already in P's cache.
6. Update P's cache to include the remaining entries, by firstly using empty cache slots (if any), and secondly replacing entries among the ones originally sent to Q.

On reception of a shuffling request, peer Q decides to accept it or not. A shuffle request is only rejected by peers that have placed a request to shuffle but have not yet received a response. If *Q* accepts the request, it randomly selects a subset of its own neighbors, of size no more than *l*, sends it to the initiating node, and executes steps 5 and 6 to update its own cache accordingly.

You will receive from the TA a template class containing instructions of how to implement this protocol in PeerSim. You can chose to follow the instructions or implement your own version of the protocol based on the description above or the original algorithm found in [1].

Once you have implemented the protocol, you will run a set of PeerSim simulations to validate some desired properties of the produced overlay, and adjust your implementation accordingly. The set of properties that you have to validate is:

1. **Shortest Path** – The average path length is a metric of the number of hops (and hence, communication costs and time) to reach nodes from a given source. A small average path length is therefore essential for broadcasting or, generally, information dissemination applications.
2. **Clustering Coefficient** – The ratio of the existing links among the node's neighbors over the total number of possible links among them. It basically shows to what percentage the neighbors of a node are also neighbors among themselves. A high average clustering coefficient indicates higher chances of network partitioning and high number of redundant message deliveries.
3. **In-degree Distribution** – The degree of a node is the number of links it has to other nodes, in the undirected connection graph. The in-degree of a node is the number of edges ending at this node in a directed graph. The degree distribution is related to the robustness of the overlay in presence of failures and indicates how the resources are distributed across the nodes. We consider only the in-degree because the out-degree will be a fixed value equals to the cache size.

A more complete discussion about these properties can be found in [2]. Essentially, the average path length, average clustering coefficient and in-degree distribution values observed in your simulations **have to converge to values similar to the ones observed in the same simulations using an overlay connected as a random graph**.

You will also get a script containing an example of such simulations. The simulation settings are the following:

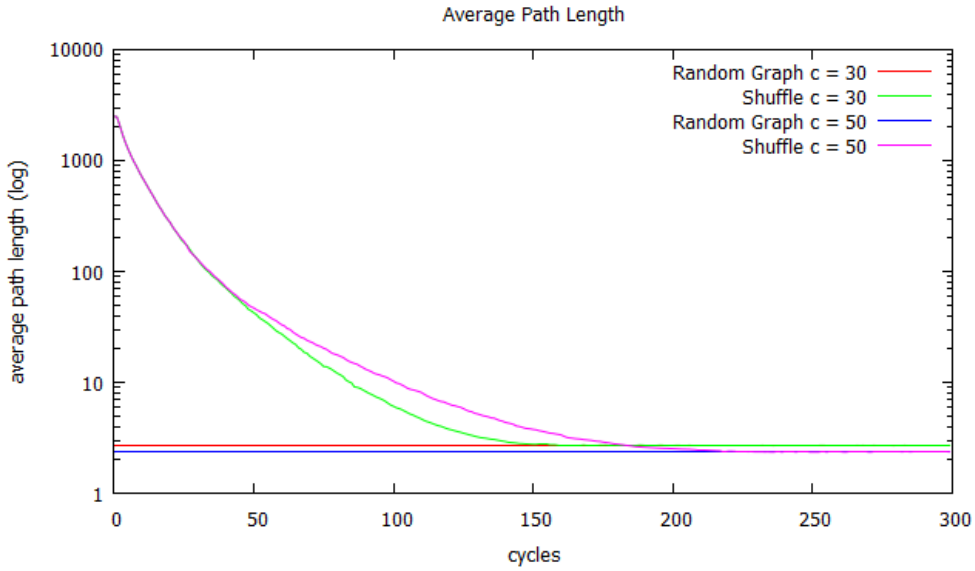
1. Your P2P system will be composed by 10.000 nodes;
2. Each simulation will run for 300 cycles, and every cycle has 1000 time units;
3. Assume random delays of minimum 100 and maximum 600 time units, and no packet loss;
4. The *shuffle length* is fixed and equals to 8;
5. You should run the simulations using two *cache size* setting: 30 and 50;
6. You should bootstrap the simulations using two topology setting: Ring and Star:
    a. Use the *WireRingLattice* and *WireStar* classes of the simulator to build these topologies as shown in the provided script.
7. Use the *GraphStats* control class to produce the *shortest path* and *clustering coefficient* information once *every* cycle, as shown in the provided script.
    a. Use a undirected graph by setting the *undir* property to *true*;

b. Shortest path is defined by the *nl* property. Set it to 10 to get the average path length of 10 random nodes;
c. Clustering coefficient is defined by the *nc* property. Set it to 1000 to get the average coefficient from 1000 random nodes;

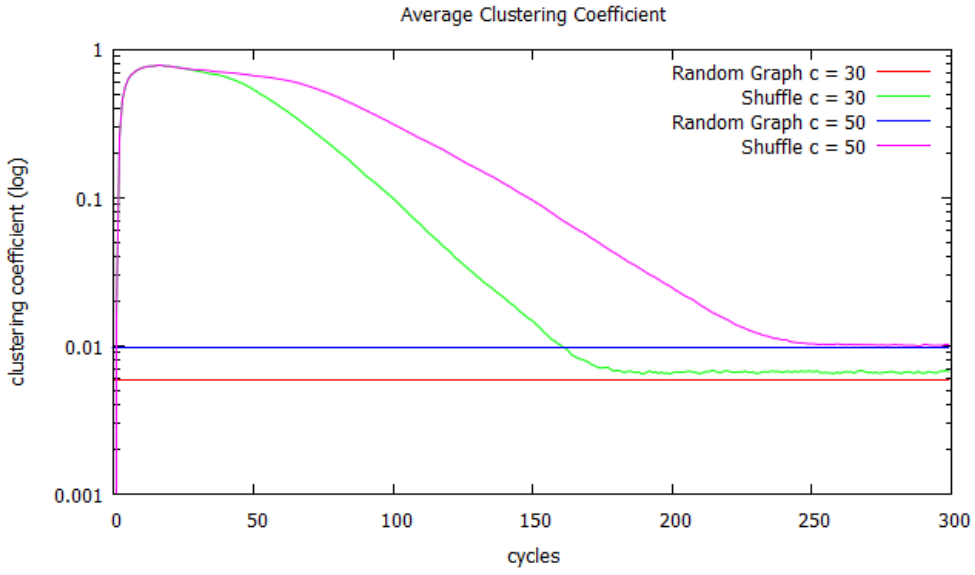
8. Use the *InDegreeObserver* control class to get the in-degree distribution information at the end of the simulation, as shown in the provided script.

The controllers *GraphStats* and In*DegreeObserver* will produce data and print it in the standard output. You can redirect the output to a file and filter it for the relevant information or change the controllers' implementation to save only the relevant data into a file. You should produce a set of graphs to show the performance of the protocol using the collected data. The graphs that you have to produce are:

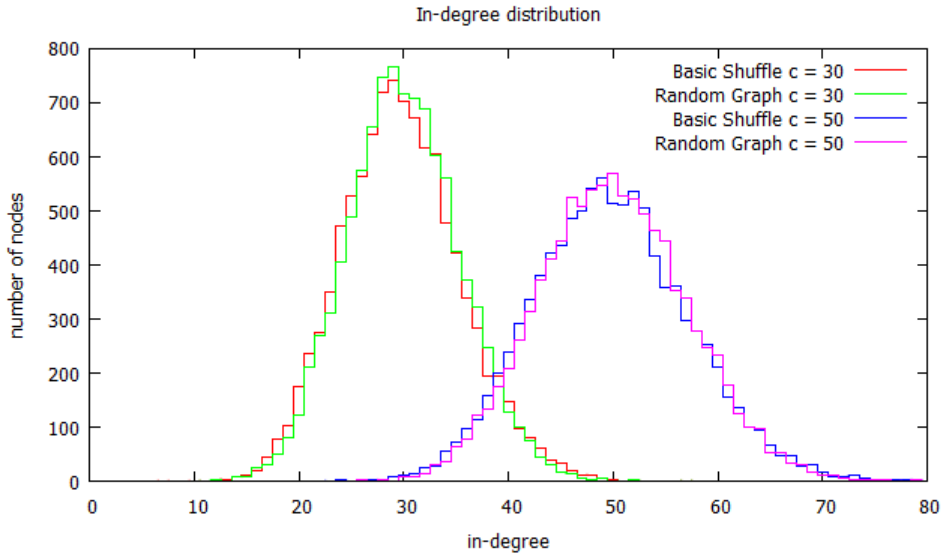
1. Average path length, as the example bellow:



2. Average clustering coefficient, as the example bellow:



3. In-degree distribution, as the example bellow:



The TA will provide you examples of scripts and you can plot the graphs using the Gnuplot tool. However, you can use any tool of your preference to plot these graphs. In total, you have to produce **six graphs**, three for each bootstrap topology type.

Development Tools:

1. *Integrated Development Environment*: Eclipse IDE for **Java EE** Developers: http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/junor
2. The PeerSim simulator: http://peersim.sourceforge.net/
3. Gnuplot: http://www.gnuplot.info/

Deliverables:

Via the Devilry system.

- The source code (only classes that you created or modified) and scripts of your simulations.
- The data files and the graphs. You should also produce a document explaining your conclusions about the simulation: Which topology do you think is better to bootstrap such a P2P system? What is the importance of the cache size?

Deadline: 23:00 on 17th. November, 2017

References:

[1] Stavrou, Angelos, Dan Rubenstein, and Sambit Sahu. "A lightweight, robust p2p system to handle flash crowds." *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*. IEEE, 2002.

[2] Voulgaris, Spyros, Daniela Gavidia, and Maarten Van Steen. "Cyclon: Inexpensive membership management for unstructured p2p overlays." *Journal of Network and Systems Management* 13.2 (2005): 197-217.