Third Assignment – Tutorial Lecture

INF5040 (Open Distributed Systems)

Tien Dat Le (dattl@ifi.uio.no)

Department of Informatics
University of Oslo

November 6, 2017

P2P System

- An application-level network on top of the Internet (overlay network)
  - Each participant contributes its own resources to the system
  - All nodes have the same functional capabilities and responsibility
  - No dependency on a central entity for administration of the system (self-organizing)
  - The effectiveness critically depends on algorithms for data placement over many nodes and for subsequent access to them
  - Unpredictable availability of processes and nodes

P2P System (2)

- Non-functional requirements
  - Global scalability
  - Load-balancing
  - Optimization for local interaction between neighbor peers (Gossiping)
  - Coping with high node and object “churn”
  - Security of data in an environment with heterogeneous trust
  - Anonymity and resilience to censorship

P2P System (3)

- Building and maintaining P2P overlays to fulfill the non-functional requirements is a challenge
- This problem is specially relevant for very large-scale and dynamic P2P systems
- In this assignment you will be introduced to the process of **developing**, **debugging** and **simulating** P2P protocol to address the aforementioned overlay challenge

Shuffling Protocol

- Overlay construction and management protocol
  - A shuffle is an exchange of a subset of neighbors between a pair of peers and can be initiated by any peer.
  - Shuffling is used to produce an overlay that is "**well-mixed**": a peer's neighbors are essentially drawn at random from the set of all peers that participate in the overlay.
  - The aim is to have the overlay displaying similar properties as the ones observed in a **random graph**.

Shuffling Protocol (2)

- Desired Properties
  - **Resilience in presence of churn**: The chances of the overlay being partitioned is minimized and no peer becomes disconnected as a result of the protocol
  - **Communication efficiency**: There is always a communication path between any pair of peers and the length of this path should be minimized
- A random graph exhibits these properties, therefore the protocol should produce overlays that are similar to random graphs.

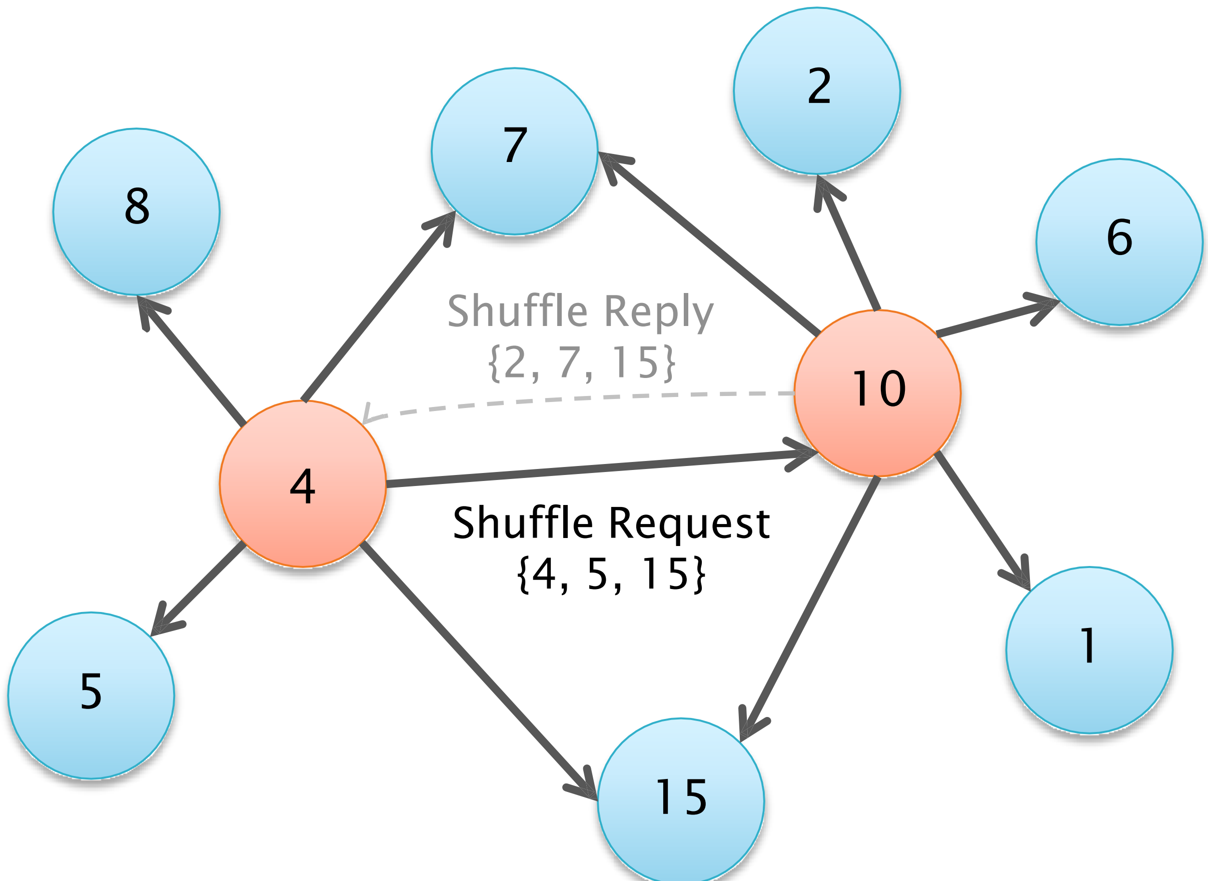
Shuffling Protocol (3)

- Basic Shuffling Protocol, initiated by a peer *P* :

1) Select a random subset of ***l*** neighbors (1 ≤ ***l*** ≤ ***c***) from ***P'***s cache, and a random peer, ***Q***, within this subset.
   - Where ***l*** is a system parameter, called *shuffle length* and ***c*** is the size of *P'*s neighbors cache.
2) Remove ***Q'***s entry from ***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) 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***.

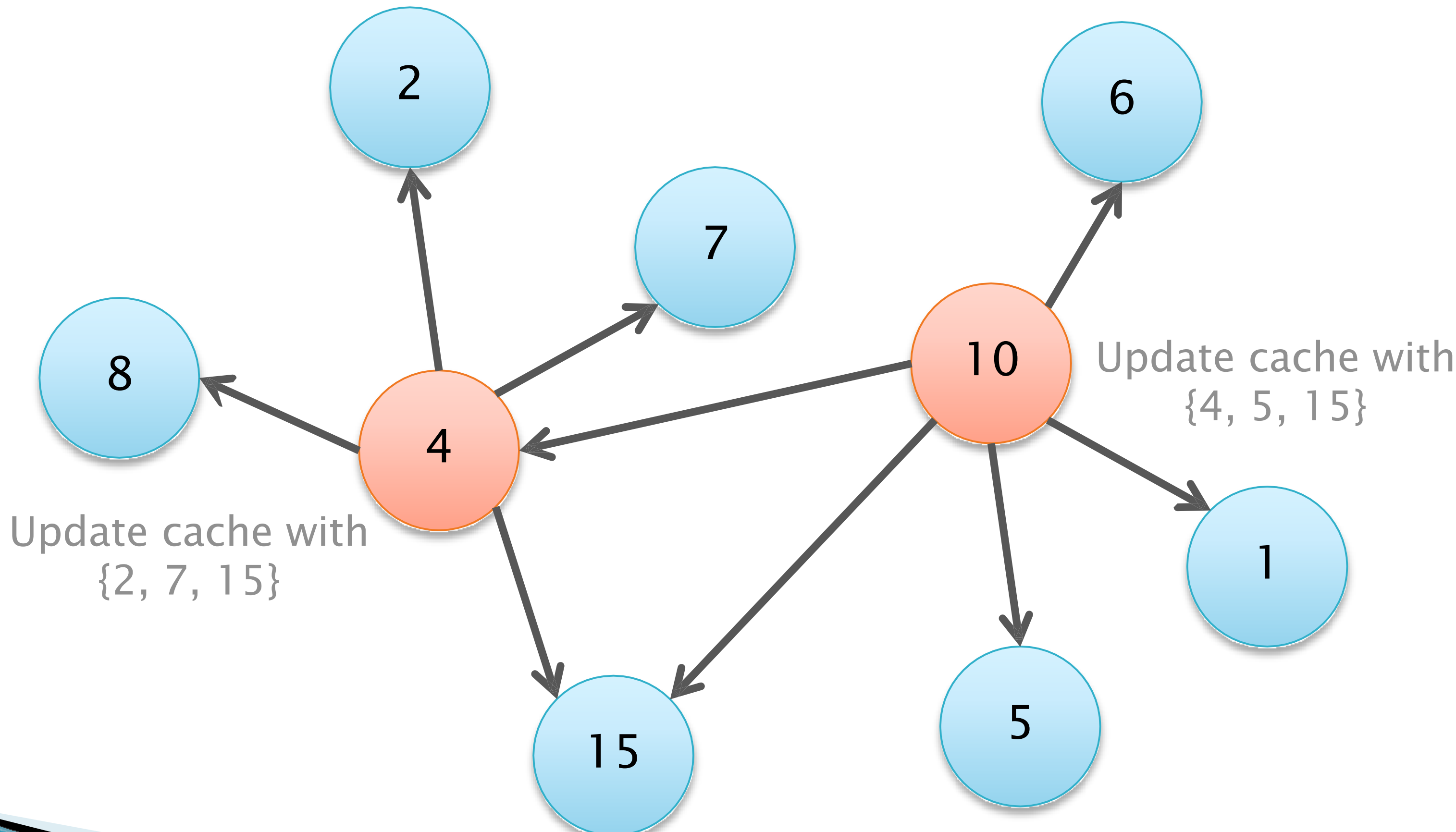
Shuffling Protocol (4)

- Basic Shuffling Protocol, example:



Shuffling Protocol (4)

- Basic Shuffling Protocol, example:



PeerSim

- PeerSim is a simulator for P2P systems
  - Supports dynamicity and very large scale systems
- Composed of a **cycle-based** and a **event-based** simulation engine
  - Cycle-based use simplifying assumptions, such as ignoring the details of the transport layer in the communication protocol stack
  - Event-based supports scheduling of events and transport layer simulation as well (transport delays)

PeerSim (2)

- Download and installation:
  - Download it from: http://sourceforge.net/projects/peersim/
  - It is a Java project described in a Ant build file
  - You can use Ant directly, or you can create a new "*Java Project from Existing Ant Buildfile*" in Eclipse.
- You can find some tutorials at: http://peersim.sourceforge.net/#docs
- Run PeerSim by using the main class **Simulator** in the peersim package, with a simulation script as argument

Simulation

- Simulation script (Event-based)

```
SIZE 10000

CYCLE 1000
CYCLES 300

MINDELAY 10
MAXDELAY 60

random.seed 1234567890
network.size SIZE
simulation.endtime CYCLE*CYCLES
simulation.logtime CYCLE
```

Number of time units in a cycle

Number of cycles

Seed for the simulator's random number generator

The size of the network in number of nodes

When the simulation ends

When the logging occurs

Simulation (2)

- Simulation script (Event-based)

```
protocol.tr UniformRandomTransport
{
   mindelay (CYCLE*MINDELAY)/100
   maxdelay (CYCLE*MAXDELAY)/100
}
protocol.gossip example.gossip.BasicShuffle
{
   cacheSize 30
   shuffleLength 8
   step CYCLE
   transport tr
   period CYCLE
}
init.wire WireStar
{
   protocol gossip
}
```

Name of the class implementing the protocol

Protocol parameters

Object that initializes the connections between peers for the protocol. In this case it creates a star topology

Simulation (3)

- Simulation script (Event-based)

```
init.sch CDScheduler
{
   protocol gossip
}
control.degree example.reports.DegreeObserver
{
   protocol gossip
   step CYCLE
   starttime 299000
   endtime 300000
}
control.graphPL GraphStats
{
   protocol gossip
   step CYCLE
   undir true
   nl 10
}
```

Initializes the event-based scheduler that will call the *nextCycle* method in the protocol implementation once every cycle

Control implementing an observer that will collect information about the in-degree distribution and display it at the end of the simulation

Control that collects information about path length and clustering and print it once every cycle

Simulation (4)

- Protocol implementation

Indicates that instances of this class can form networks of linked nodes

```
public class BasicShuffle  implements Linkable,
EDProtocol, CDProtocol{
```

This protocol follows the event-driven model

This protocol also follows the cycle-driven model

```
    public void nextCycle(Node node, int protocolID) { }
```

This method is called by the simulator scheduler once every cycle

```
    public void processEvent(Node node, int pid, Object
        event) { }
}
```

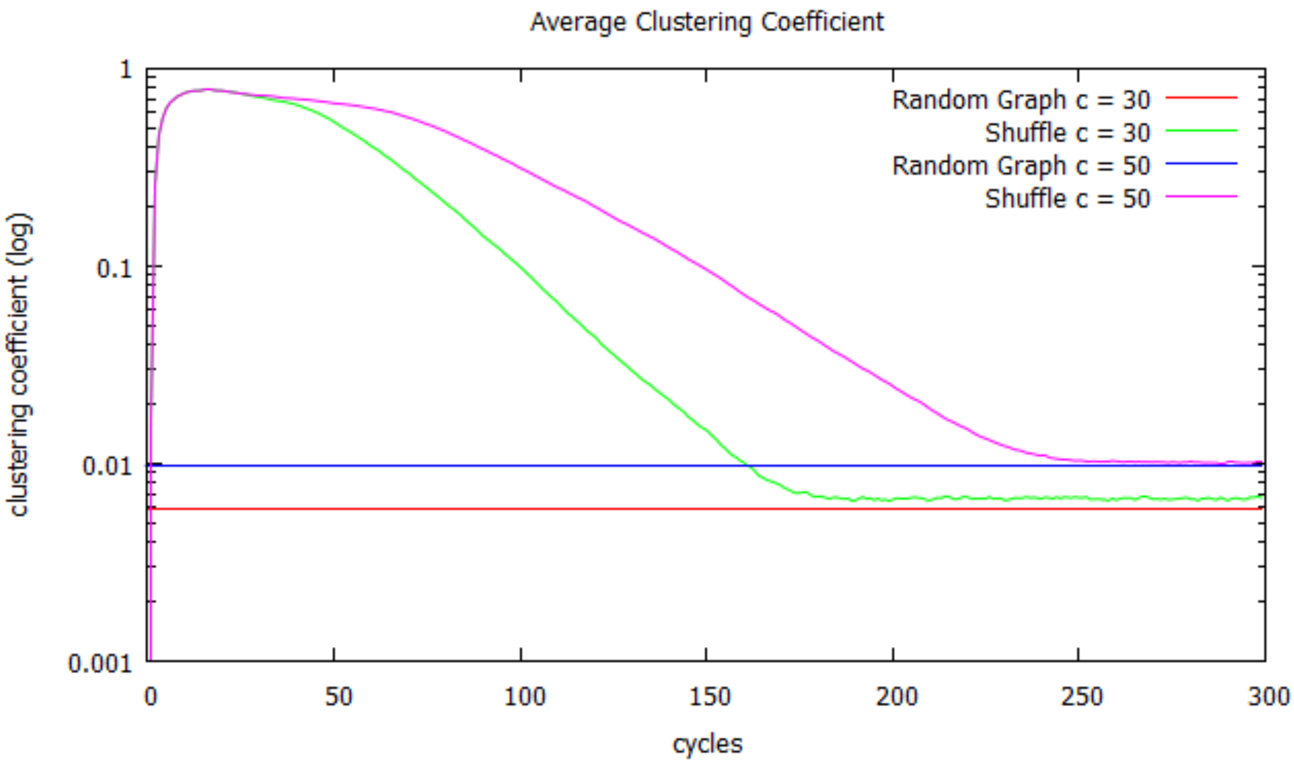
This method is called by the scheduler every time an event occurs, such as a node receiving a message.

Results

- Simulation Results:
  - **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.
  - **Clustering Coefficient** –It basically shows to what percentage the neighbors of a node are also neighbors among themselves.
  - **In-degree Distribution** –The in-degree of a node is the number of edges ending at this node in a directed graph.
- The values observed in your simulations have to converge to values similar to the ones observed in the same simulations using a overlay connected as a random graph

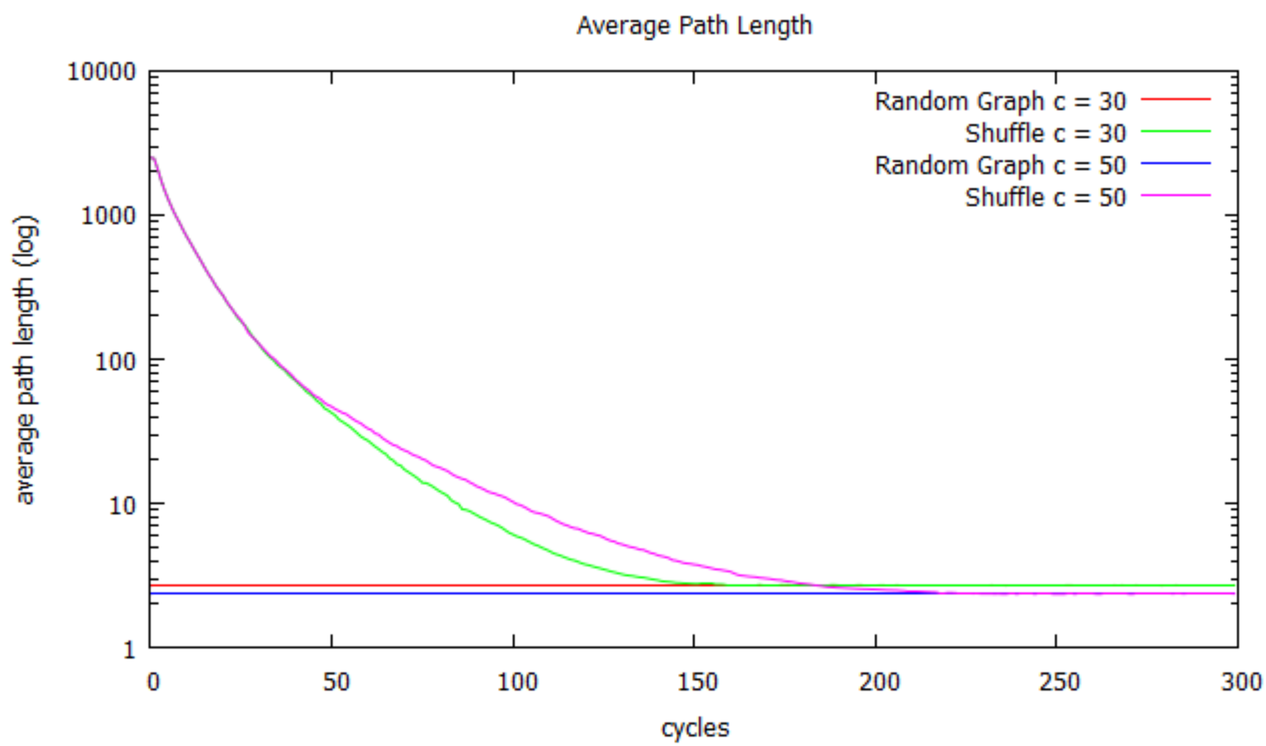
Results (2)

- Simulation Results – Graphs:



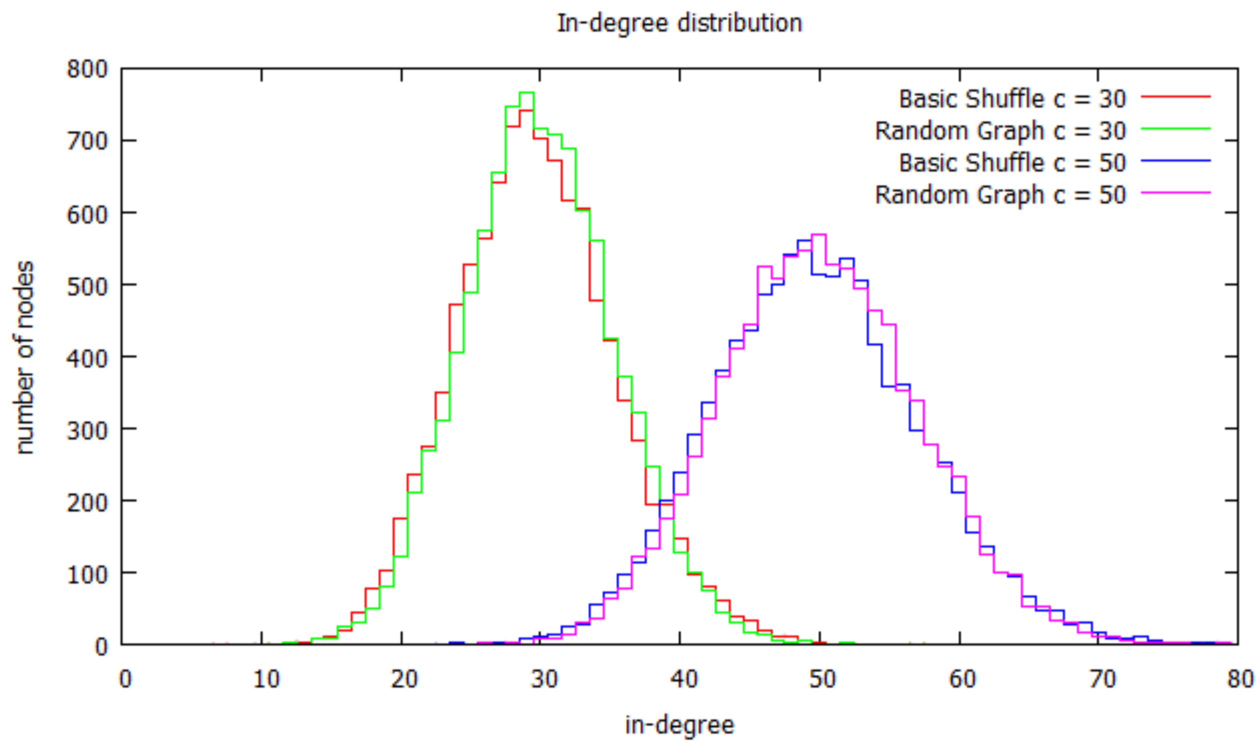
Results (3)

- Simulation Results – Graphs:



Results (4)

- Simulation Results – Graphs:



Plotting

- Simulation Results – Graphs:
  - You can use any plotting tool to produce these graphs
  - Example of gnuplot scripts will be provided:

```
set title "Average Clustering Coefficient (Ring Topology)"
set xlabel "cycles"
set ylabel "clustering coefficient (log)"
set key right top
set logscale y
plot "ccRandom30.txt" title 'Random Graph c = 30' with lines, \
        "cc30.txt" title 'Shuffle c = 30' with lines, \
        "ccRandom50.txt" title 'Random Graph c = 50' with lines, \
        "cc50.txt" title 'Shuffle c = 50' with lines
```

Implementation Kit

- Download the PeerSim kit from the course page. It contains:
  - Template of the Basic Shuffle implementation
  - Examples of PeerSim scripts using the shuffle protocol
  - Examples of Gnuplot scripts to produce the required graphs
  - Implementation of an observer to produce in-degree distribution information
- Add these artifacts to your own PeerSim installation

Deliverables

- The source code
  - only classes that you created or modified
- Scripts of your simulations.
- The data files and the graphs.
- 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: November 17, 2017**