Complex Adaptive Systems
Course's Project Guidelines

Stefano Arteconi

Dipartimento di Scienze
dell'Informazione
Università di Bologna

Outline

- Project Specification
  - Drop and rewire mechanisms
  - Project specifics
- Peersim guidelines
  - Writing a CDProtocol
  - Writing a Control
  - Writing a Configuration File

Project Specification

SLAC Pseudocode

Generic node

Perform application task (APPLICATION layer)

Get utility

Periodically do

- Choose a random node (NEWSCAST layer)
- Compare utilities
- Node with lower utility
  - Copies winner's strategy
  - **DROP: leaves current neighborhood (SLAC layer)**
  - **REWIRE: Joins winner's neighborhood (SLAC layer)**

Mutate

What You Are Asked to Do: New Drop and Rewire Mechanisms

- Define new way of dropping and rewiring links
  - Rewire probability similar to drop probability
  - Link to some 2-hops neighbor
  - ...
- This is totally up to you!!!
- The global emergent mechanism should be understood
  - What we want (cooperation)
  - Basic requirements (communication between cooperating nodes)
- Local rules defined accordingly

Drop and Rewire Parametrization

- Drop and Rewire mechanisms should be parametrized
  - Example: drop probability in SLACER
- Evaluate different drop and rewire parameter settings
  - Dependent on the drop and rewire mechanism itself

Drop and Rewire effects

- Define new way of dropping and rewiring links
- Study dropping and rewiring effects
  - Cooperation
    - Level
    - Time
  - Topology
    - CC
    - APL
  - Mixed
    - LCC
    - CCP
    - CCPL

Network Settings Effects

- Examine initial settings effects on performances
  - Network size
  - Initial topology
    - Use existing classes (*WireKOut*, *WireRingLattice*, *WireStar*, ...)
  - Churning
    - Use existing class *DynamicNetwork*

Beware not to Cheat!

- Everything is based on local knowledge
- Other nodes' state known only when interacting with them
- "Cheating" examples
  - Choose nodes directly from *Network* structure
    - NO GLOBAL KNOWLEDGE!
    - Only local views (newscast and slacer) available for each node
  - Link only to cooperative nodes
    - Other nodes states not known during rewiring
    - Possibly game not completely known by players (i.e. nodes are not aware of the payoff table values)

Project package

- Class package *casproject* provides
  - *Slacer*
    - Implements Slacer and PD application
    - To be extended
    - *drop()* and *rewire()* abstract methods to be implemented
  - *CCPObserver*
    - Evaluates LCC, CCP and CCPL
  - *StrategyObserver*
    - Counts number of cooperating and defecting nodes

Additional Classes

- If you need some additional mechanism feel free to implement and use it!
  - New specific observers
    - i.e. measuring stats related to nodes' utilities
  - New protocol layer
    - If overlay topologies other than the ones provided by Newscast and Slacer are needed

Peersim Guidelines

Implementing new classes

- All the new code written should be included in classes implementing one (or more) of the main peersim interfaces
  - *CDProtocol*
  - *Linkable*
  - *Control*
- Examples about how to implement these interfaces will now be given

Implementing *CDProtocol* Interface

- `peersim.cdsim.CDProtocol` provides a single method to describe protocol actions
  - `void nextCycle(Node node, int protocolID);`
    - `node` is the current running node
    - `protocolID` is the current protocol identifier in the current node protocol stack
- In `nextCycle()` any action performed by each node at each simulation cycle should be implemented
- `clone()` method should be extended to clone all the defined complex structures

Implementing *Linkable* Interface

- `peersim.core.Linkable` provides abstract methods for node view management
  - `public int degree();`
  - `public Node getNeighbor(int i);`
  - `public boolean addNeighbor(Node neighbour);`
  - `public boolean contains(Node neighbor);`
- Usually to manage node view just means working with an array

IdleProtocol Class

- `peersim.core.IdleProtocol` implements both *CDProtocol* and *Linkable*
- Provides view management for free
- *CDProtocol* implementation
  - Do Nothing!
- *Linkable* implementation
  - Node views are represented through arrays
  - Unlimited viewsize

Implementing *Control* Interface

- `peersim.core.Control` provides a single method to describe global network control actions
  - `boolean execute();`
- In `execute()` any action performed at each simulation cycle for global management should be implemented
- No explicit distinction between initialization, dynamicity and observation

Initialization, Dynamicity, Observation

- Initialization, Dynamicity, and Observation are all implemented through *Control* interface
- Initialization differs from dynamicity and observation
  - It should only be executed at simulation start
  - Declared in configuration file (shown later)
- Observation vs. Dynamicity
  - Observation should not modify nodes' state, Dynamicity could
  - This is a logical distinction not taken into account in peersim

Defining Configuration Parameters

- It is possible that components need parameters to be passed by configuration file
  - *CDProtocol* related parameters
  - It is required for *Control* components to specify the protocol they are related to
- `Peersim.config.Configuration` provides utilities to read parameters from file
  - Numeric values
  - Check for (un)defined values

CDProtocol Implementation Example

```
//configuration parameter definition
public static final String PAR_VIEWSIZE = "viewsize";
public static final String PAR_MSGSIZE = "msgsize";

Public class myProtocol extends IdleProtocol implements
CDProtocol {
public myProtocol(String prefix) {
    //parameters value retrieving from cofiguration file
    viewsize=Configuration.getInt(prefix+"."+PAR_VIEWSIZE);
    msgsize =Configuration.getInt(prefix+"."+PAR_MSGSIZE);
}

public Object clone() throws CloneNotSupportedException{
    super.clone();
    //complex structures cloned here
}

public void nextCycle(Node node, int protocolID) {
    // protocol behaviour here
}
}
```

Control Implementation Example

```
public static final String PAR_PROT = “protocol";
private IncrementalStats stats = new IncrementalStats();
private String name;

public class myObserver implements Control{
public myObserver(String prefix) {
    pid = Configuration.getPid(prefix + "." + PAR_PROT);
    stats = new IncrementalStats();
    name = prefix;
}

public boolean execute() {
    for(int i = 0 ; i < Network.size() ; i++){
        // pick each node and extract sensible data
        myProtocol p =(myProtocol)Network.get(i).getProtocol(pid);
        stats.add(p.value)
        ...
    }

    System.out.println(name+” “+stats.getAverage());
    return false;
}
```

Writing the configuration file - *Simulator*

```
simulation.cycles 100
Control.shf Shuffle
network.size 1000
network.maxSize 100000

protocol.0 example.newscast.SimpeNewscast
protocol.0.cache 20

protocol.1 casproject.<YOURSLACEREXTENDEDCLASS>
protocol.1.linkable 0
protocol.1.<YOURADDITIONALPARAMETERS> <VALUE>

init.0 peersim.dynamics.WireKOut
init.0.protocol 0
init.0.k 20

init.1 casproject.SlacerInitializer
init.1.protocol 1
init.1.quantity 0.5

init.2 peersim.dynamics.WireOut
init.2.protocol 1
init.2.k 20

control.1 casproject.CCPObserver
control.1.protocol 1
control.1.nl 100
```

Writing the configuration file
RangeSimulator

```
simulation.cycles 100
Control.shf Shuffle

range.netsize NETSIZE;1000,10000
range.viewsize VIEWSIZE;15,30

network.size NETSIZE
network.maxSize 100000

protocol.0 example.newscast.SimpeNewscast
protocol.0.cache VIEWSIZE

protocol.1 casproject.<YOURSLACEREXTENDEDCLASS>
protocol.1.linkable 0
protocol.1.<YOURADDITIONALPARAMETERS> <VALUE>

init.0 peersim.dynamics.WireKOut
init.0.protocol 0
init.0.k 20

...
```

Working with Peersim

Getting Peersim

- Sun Java JDK5.0 (J2SE 1.5.0)
- Peersim P2P Simulator (http://peersim.sourceforge.net/): the CVS version is recommended:
  - cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/peersim login
  - cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/peersim co -P peersim
- Package jep (Java Expression Parser)
- Gnuplot Software (ver. >= 3.7.x) to plot data

Compiling and Launching the Simulation

- Peersim can be compiled using Makefile or ant (both provided by CVS version)
- Single run
  - `java -cp .:jep-2.24.jar peersim.Simulator <configfile>`
- Multiple runs
  - `java -cp .:jep-2.24.jar peersim.RangeSimulator <configfile>`
  - Config file should end with .cfg
  - Makes a run for each parameters combination

Plotting the results

- Log files can be easily plotted using gnuplot
- The basic gnuplot command format is
  - `plot <nomefile> using <n1>:<n2> every <n3> with lines`
  - Plots data in <nomefile> using $n1^{th}$ and $n2^{th}$ values every n3 lines with a continuous line
- For further information about Gnuplot please refer to online documentation
  - http://www.gnuplot.info/documentation.html

If you need any help...

- Look for documentation at: http://peersim.sf.net
- My homepage: http://www.cs.unibo.it/~arteconi
- My eamil address: arteconi@cs.unibo.it
- My office: Dip. Scienze dell'Informazione - Lab C, 1st floor underground