Simulating P2P Systems with PeerSim

Peer To Peer Systems
2014/2015
Prof. Laura Ricci

Speaker: Alessandro Lulli - lulli@di.unipi.it

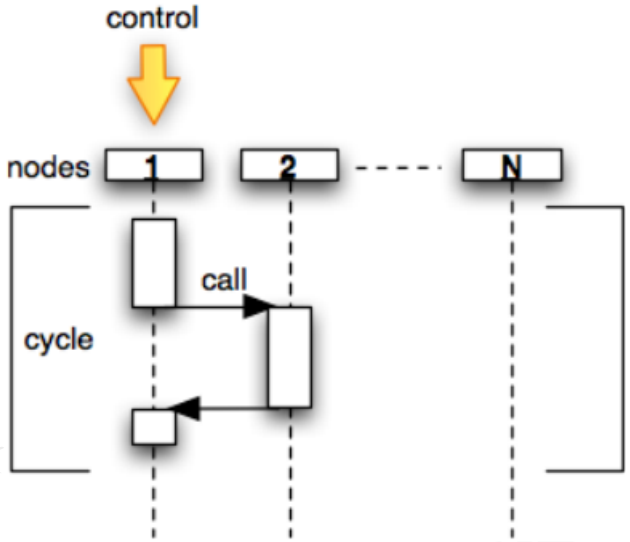
What is PeerSim

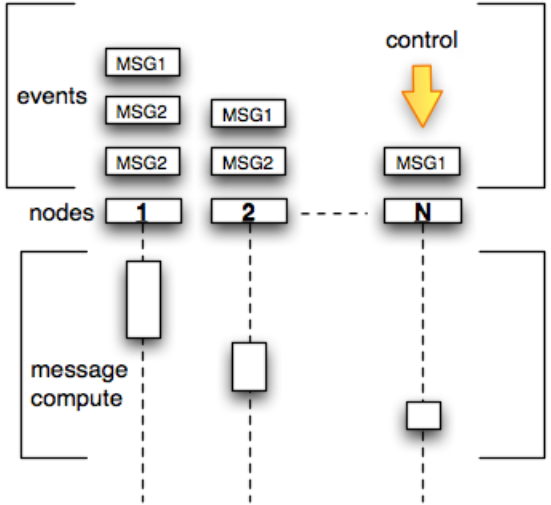
- PeerSim is an open source P2P simulator available at:
  - http://peersim.sourceforge.net/
- Java
- high scalability (up to 1 million nodes)
- highly configurable

- acknowledgements, part of this material is taken from :
  - Alberto Montresor and Gianluca Ciccarelli “Peersim informal introduction”
  - Moreno Marzolla “Simulating overlay network with PeerSim”
  - Andrea De Salve “Simulating with PeerSim” 2013/2014

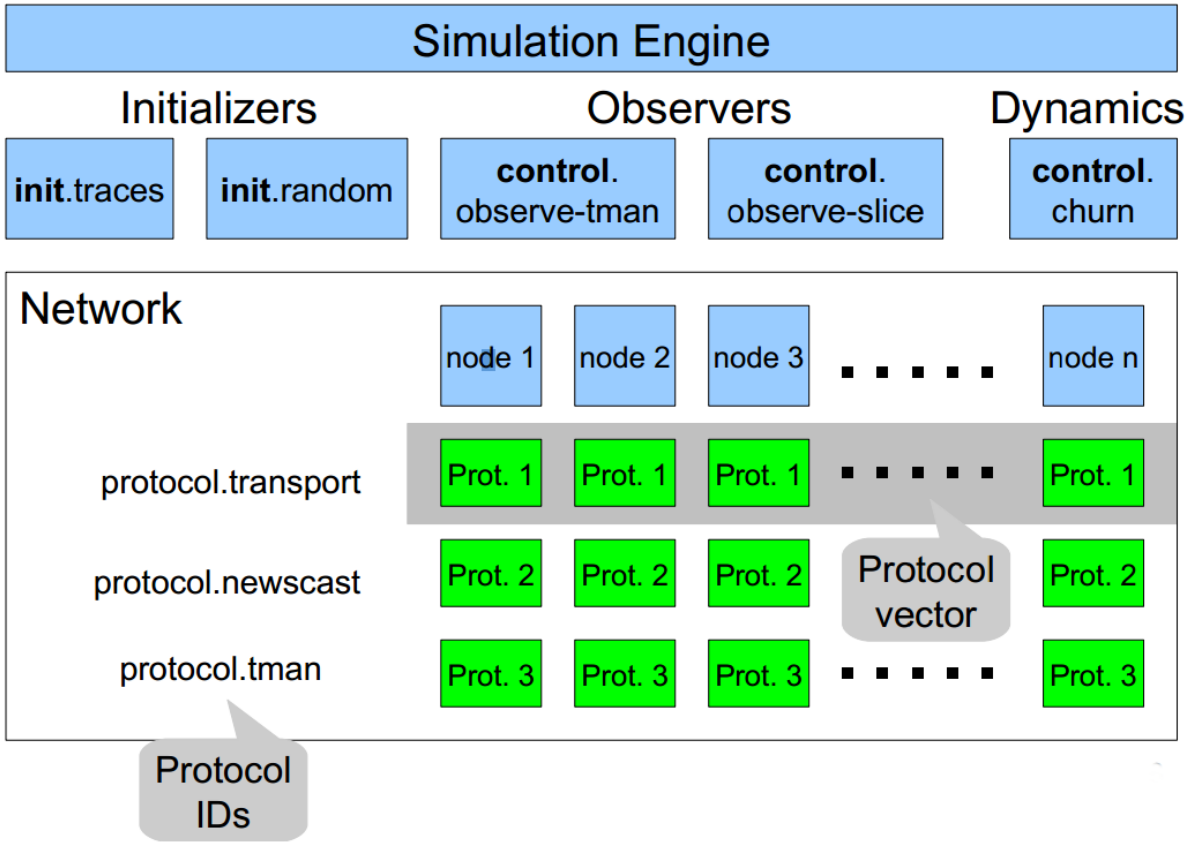
Simulation engine

- cycle-driven (CD)
  - no messages
  - no transport
- event-driven (ED)
  - message based
  - realistic transport



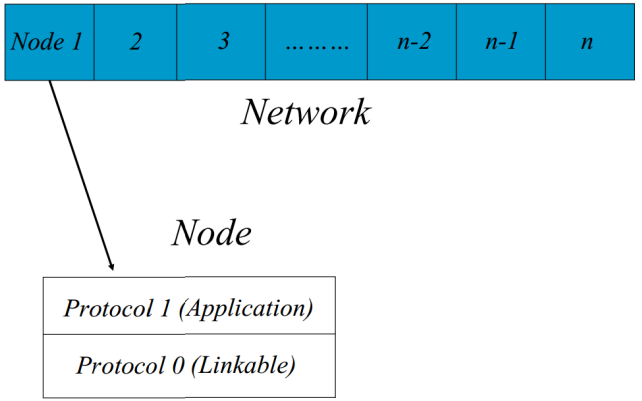


Components

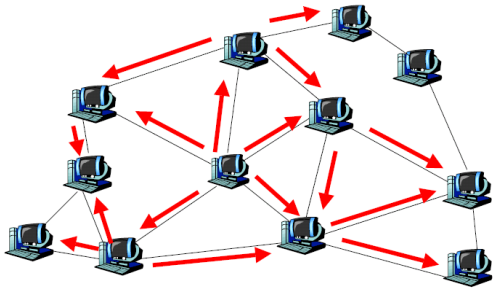

Simulation Engine
Initializers
Observers
Dynamics
init.traces
init.random
control. observe-tman
control. observe-slice
control. churn
Network
node 1
node 2
node 3
node n
protocol.transport
protocol.newscast
protocol.tman
Prot. 1
Prot. 2
Prot. 3
Protocol vector
Protocol IDs

Network representation

- Network is a global array which contains all the network nodes
- Node state and actions are described through a stack of protocols
  - Node node = Network.getNode(nId)
  - node.getProtocol(pId)
- Linkable is the interface used to access and manage node's view



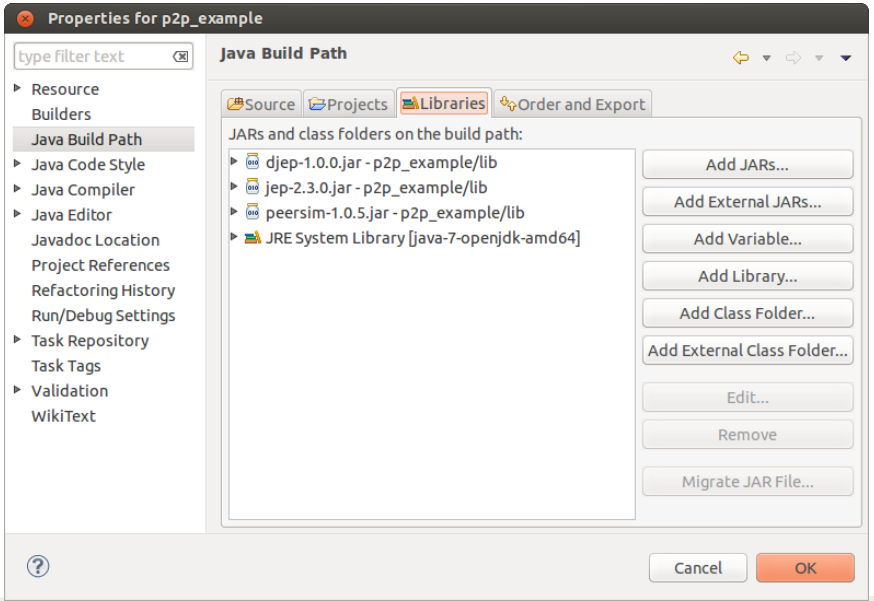
Running Example: Flooding



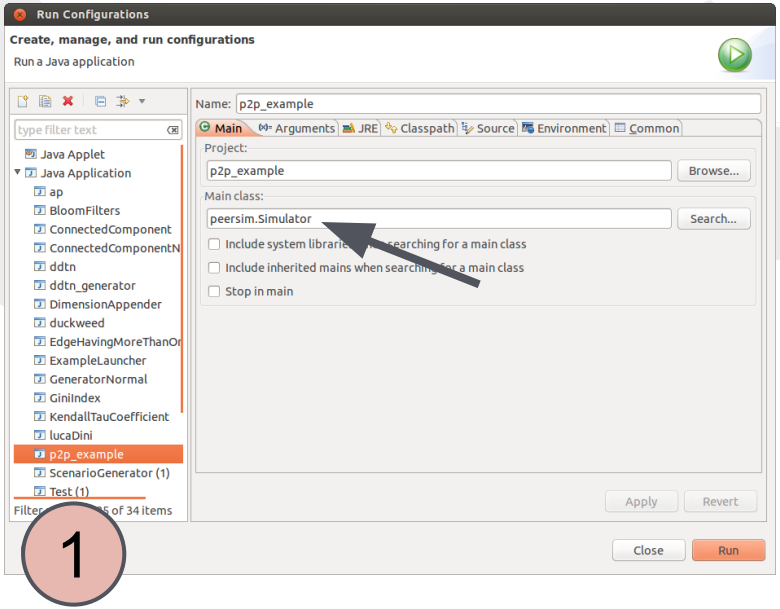
- problem: node lookup with flooding
- simulation engine: cycle driven implementation
- protocols:
  - an example of Linkable protocol
  - an example of cycle driven protocol
- initializers:
  - choose the node *v* starting the search
  - choose the node *w* to search
  - choose the maximum number of flooding step
- observer: count nodes involved in flooding

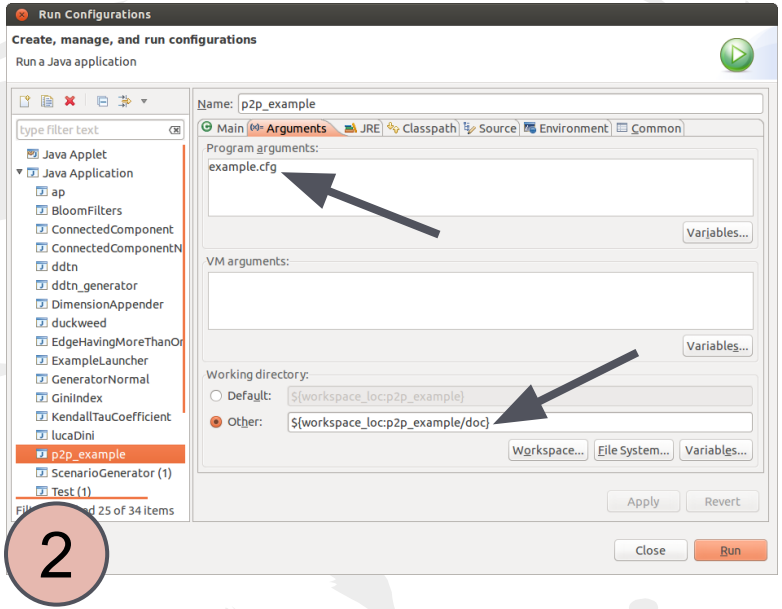
PeerSim with Eclipse (I)

- download zip file from: www.alessandrolulli.com/uni/p2p/p2p_class.zip
- from the Project properties add all the jar under **lib** folder



PeerSim with Eclipse (II)





- set the main class *peersim.Simulator*

- set the scenario file name: *example.cfg*
- set the working directory

Code: Protocol

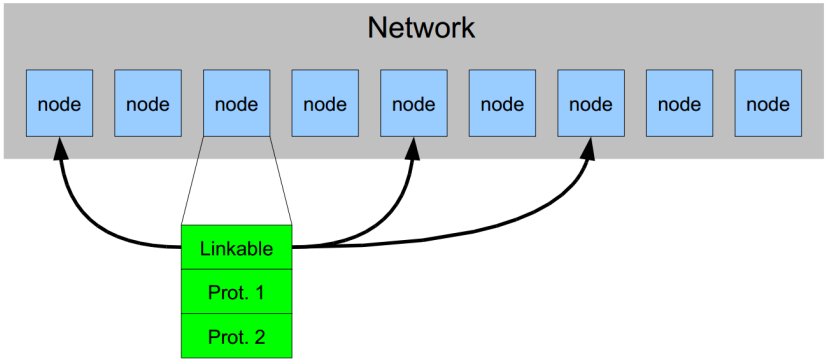
SHIFT + CTRL + H
type “Protocol”
ENTER

```
public interface Protocol extends Cloneable
{

/**
 * Returns a clone of the protocol. It is important to pay attention to
 * implement this carefully because in peersim all nodes are generated by
 * cloning except a prototype node. That is, the constructor of protocols is
 * used only to construct the prototype. Initialization can be done
 * via {@link Control}s.
 */
public Object clone();
```

Main interfaces: Linkable

- used to manage node's view
- typical actions:
  - add neighbour
  - get neighbour
  - node's degree
  - remove neighbour



Code: Linkable

SHIFT + CTRL + H
type “Linkable”
ENTER

```
public interface Linkable extends Cleanable {


    /**
    * Returns the size of the neighbor list.
    */
    public int degree();

    /**
    * Returns the neighbor with the given index. The contract is that
    * listing the elements from index 0 to index degree()-1 should list
    * each element exactly once if this object is not modified in the
    * meantime. It throws an IndexOutOfBounds exception if i is negative
    * or larger than or equal to {@link #degree}.
    */
    public Node getNeighbor(int i);

    /**
    * Add a neighbor to the current set of neighbors. If neighbor
    * is not yet a neighbor but it cannot be added from other reasons,
    * this method should not return normally, that is, it must throw
    * a runtime exception.
    * @return true if the neighbor has been inserted; false if the
    *     node is already a neighbor of this node
    */
    public boolean addNeighbor(Node neighbour);

    /**
    * Returns true if the given node is a member of the neighbor set.
    */
    public boolean contains(Node neighbor);
```

Example: LinkableExample

```
public class LinkableExample implements Linkable, Protocol
{
    protected List<Node> _neighbors;

    public LinkableExample(final String prefix_)
    {
        _neighbors = new ArrayList<Node>();
    }

    public List<Node> getNeighboar()
    {
        return _neighbors;
    }

    @Override
    public Object clone()
    {
        LinkableExample ip = null;
        try
        {
            ip = (LinkableExample) super.clone();
        } catch (final CloneNotSupportedException e)
        {
            System.out.println("HELP");
        }
        ip._neighbors = new ArrayList<Node>();
        return ip;
    }
```

```
@Override
public boolean addNeighbor(final Node n)
{
    if ((n != null) && (false == _neighbors.contains(n)))
    {
        _neighbors.add(n);
        return true;
    } else
    {
        return false;
    }
}

@Override
public Node getNeighbor(final int i)
{
    return _neighbors.get(i);
}

public Node lookup(final long id_)
{
    for(final Node n : _neighbors)
    {
        if(n.getID() == id_)
        {
            return n;
        }
    }

    return null;
}
```

SHIFT + CTRL + H
type “LinkableExample”
ENTER

Main interfaces: CDProtocol

- is used to define cycle-driven protocols, that is the actions performed by each node at each cycle
- each node can run more than one protocol
- protocols are executed sequentially

Code: CDProtocol

```
24 /**
25  * Defines cycle driven protocols, that is, protocols that have a periodic
26  * activity in regular time intervals.
27  */
28 public interface CDProtocol extends Protocol
29 {
30
31 /**
32  * A protocol which is defined by performing an algorithm in more or less
33  * regular periodic intervals.
34  * This method is called by the simulator engine once in each cycle with
35  * the appropriate parameters.
36  *
37  * @param node
38  *          the node on which this component is run
39  * @param protocolID
40  *          the id of this protocol in the protocol array
41  */
42 public void nextCycle(Node node, int protocolID);
43
44 }
```

SHIFT + CTRL + H
type “CDProtocol”
ENTER

Example: CDProtocolExample

```
@Override
public void nextCycle(final Node node_, final int protocolID)
{
    final CDProtocolExample cdProtocol = ((CDProtocolExample)node_.getProtocol(protocolID));
    final LinkableExample linkableProtocol = ((LinkableExample)node_.getProtocol(_linkableProtocolId));

    if(cdProtocol.isNodeToFind())
    {
        final Node nodeFounded = linkableProtocol.lookup(_nodeToFindId);

        if(nodeFounded != null)
        {
            cdProtocol.nodeFounded(true);
        } else if(_propagationStep > 0)
        {
            for(int i = 0 ; i < linkableProtocol.degree() ; i ++)
            {
                final Node neighbour = linkableProtocol.getNeighbor(i);
                final CDProtocolExample neighbourCdProtocol = ((CDProtocolExample)neighbour.getProtocol(protocolID));

                neighbourCdProtocol.propagateSearch(cdProtocol, _nodeToFindId, _propagationStep - 1);
            }
        } else
        {
            cdProtocol.nodeFounded(false);
        }
    }

    if(cdProtocol.isToNotify())
    {
        cdProtocol.notifyNodes(_nodeFounded);
    }
}
```

SHIFT + CTRL + H
type “CDProtocolExample”
ENTER

Observers

- Control Interface
- Perform periodic actions
- Observe/Change the state of the system
- Send messages
- (E.g. print the neighbors of each node)

```
public interface Control
{

/**
 * Performs arbitrary modifications or reports arbitrary information over the
 * components.
 * @return true if the simulation has to be stopped, false otherwise.
 */
public boolean execute();
```

SHIFT + CTRL + H
type “Control”
ENTER

Example: ControlExample

```
private final int _cdProtocolPid;

public ControlExample(final String prefix_)
{
    _cdProtocolPid = Configuration.getPid(prefix_ + ".cdProtocol");
}

@Override
public boolean execute()
{
    final boolean isSearchCompleted = isSearchCompleted();

    if(isSearchCompleted)
    {
        System.out.println("Node partecipating: "+ countNodePartecipating());
        reset();
    }

    return isSearchCompleted;
}

public int countNodePartecipating()
{
    int nodePartecipating = 0;

    for(int i = 0 ; (i < Network.size()) ; i ++)
    {
        final CDProtocolExample cdProtocol = ((CDProtocolExample)Network.get(i).getProtocol(_cdProtocolPid));

        if(cdProtocol.isNodePartecipating())
        {
            nodePartecipating ++;
        }
    }

    return nodePartecipating;
}

public boolean isSearchCompleted()
{
    boolean completed = true;

    for(int i = 0 ; (i < Network.size()) && completed ; i ++)
    {
        final CDProtocolExample cdProtocol = ((CDProtocolExample)Network.get(i).getProtocol(_cdProtocolPid));

        completed &= cdProtocol.isSearchCompleted();
    }

    return completed;
}
```

- *execute:* when search is completed:
  - check all node participating
  - print the number of nodes participating

SHIFT + CTRL + H
type “ControlExample”
ENTER

Initializers and example: InitExample

- executed at the beginning of the simulation
- Control interface
- Initialize protocols

```
public class InitExample implements Control
{
    private final int _cdProtocolPid;
    private final int _startingNode;
    private final int _searchNode;
    private final int _maxPropagation;

    public InitExample(final String prefix_)
    {
        _cdProtocolPid = Configuration.getPid(prefix_ + ".cdProtocol");

        _startingNode = Configuration.getInt(prefix_ + ".nodeStart");
        _searchNode = Configuration.getInt(prefix_ + ".nodeSearch");
        _maxPropagation = Configuration.getInt(prefix_ + ".maxPropagation");
    }

    @Override
    public boolean execute()
    {
        final CDProtocolExample cdProtocol = ((CDProtocolExample)Network.get(_startingNode).getProtocol(_cdProtocolPid));

        cdProtocol.propagateSearch(cdProtocol, _searchNode, _maxPropagation);

        return false;
    }
}
```

SHIFT + CTRL + H
type “InitExample”
ENTER

Flooding Configuration

```
1
2 random.seed 1234567890
3
4 simulation.cycles 100
5
6 network.size 1000
7
8 protocol.linkableExample cycleDrivenFlooding.LinkableExample
9
10 protocol.cdExample cycleDrivenFlooding.CDProtocolExample
11 protocol.cdExample.linkable linkableExample
12
13 init.wire cycleDrivenFlooding.WireRandom
14 init.wire.protocol linkableExample
15 init.wire.degreeMin 1
16 init.wire.degreeMax 3
17
18 init.startSearch cycleDrivenFlooding.InitExample
19 init.startSearch.cdProtocol cdExample
20 init.startSearch.nodeStart 2
21 init.startSearch.nodeSearch 8
22 init.startSeacrh.maxPropagation 3
23
24 control.check cycleDrivenFlooding.ControlExample
25 control.check.cdProtocol cdExample
26
```

- random.seed: to replicate the test
- simulation.cycle: to limit the number of cycles
- network.size: to define the number of nodes

- protocol.*x*: define a new protocol
- protocol.*x*.var: define the value for a variable for protocol *x*

How to read a configuration variable

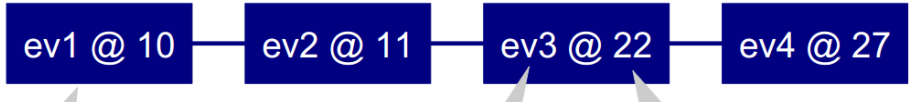
```
public InitExample(final String prefix_)
{
    _cdProtocolPid = Configuration.getPid(prefix_ + ".cdProtocol");

    _startingNode = Configuration.getInt(prefix_ + ".nodeStart");
    _searchNode = Configuration.getInt(prefix_ + ".nodeSearch");
    _maxPropagation = Configuration.getInt(prefix_ + ".maxPropagation");
}
```

- ***prefix_*** is the substring relative to this class (e.g. init.startSearch)
- ***Configuration.getPid*** to get the protocol id
- Configuration.getInt to get Int value from config
- additional parameter to set a ***default value***

Main interfaces: EDProtocol

- in the event-driven paradigm, the simulator handles an event queue, which contains events




```
public interface EDProtocol
extends Protocol
{

    /**
    * This method is invoked by the scheduler to deliver events to the
    * protocol. Apart from the event object, information about the node
    * and the protocol identifier are also provided. Additional information
    * can be accessed through the {@link CommonState} class.
    *
    * @param node the local node
    * @param pid the identifier of this protocol
    * @param event the delivered event
    */
    public void processEvent( Node node, int pid, Object event );
```

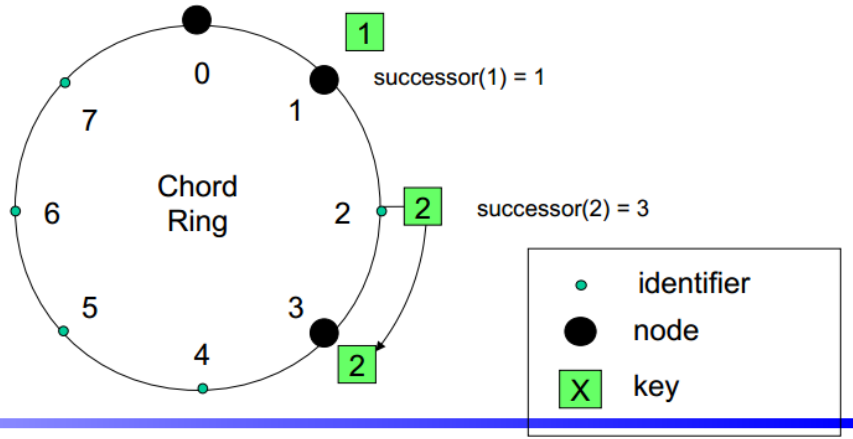
SHIFT + CTRL + H
type “EDProtocol”
ENTER

Chord

- routing:
  - log(N) hops with high probability
  - routing table size: log(N) with high probability
- self-organization
  - self-adaptation in presence of new nodes joins and voluntary/abrupt nodes leave

THE CHORD RING

- the ring includes identifiers, the arithmetic is mod 2^{160} (in the example mod 2^3)
- the key and the corresponding value are managed by the successor of the key in the clockwise ordering



Example: ChordProtocol

```java
public class ChordProtocol implements EDProtocol {

    private static final String PAR_TRANSPORT = "transport";

    private Parameters p;

    private int[] lookupMessage;

    public int index = 0;

    public Node predecessor;

    public Node[] fingerTable;

    public Node[] successorList;

    public BigInteger chordId;
```

SHIFT + CTRL + H
type “ChordProtocol”
ENTER

```java
public void processEvent(Node node, int pid, Object event) {
    // processare le richieste a seconda della routing table del nodo
    p.pid = pid;
    currentNode = node.getIndex();
    if (event.getClass() == LookUpMessage.class) {
        LookUpMessage message = (LookUpMessage) event;
        message.increaseHopCounter();
        BigInteger target = message.getTarget();
        Transport t = (Transport) node.getProtocol(p.tid);
        Node n = message.getSender();
        if (target == ((ChordProtocol) node.getProtocol(pid)).chordId) {
            // mandare mess di tipo final
            t.send(node, n, new FinalMessage(message.getHopCounter()), pid);
        }
        if (target != ((ChordProtocol) node.getProtocol(pid)).chordId) {
            // funzione lookup sulla fingertabable
            Node dest = find_successor(target);
            if (dest.isUp() == false) {
                do {
                    varSuccList = 0;
                    stabilize(node);
                    stabilizations++;
                    fixFingers();
                    dest = find_successor(target);
                } while (dest.isUp() == false);
            }
            if (dest.getID() == successorList[0].getID()
                    && (target.compareTo(((ChordProtocol) dest
                            .getProtocol(p.pid)).chordId) < 0)) {
                fails++;
            } else {
                t.send(message.getSender(), dest, message, pid);
            }
        }
    }
    if (event.getClass() == FinalMessage.class) {
        FinalMessage message = (FinalMessage) event;
        lookupMessage = new int[index + 1];
        lookupMessage[index] = message.getHopCounter();
        index++;
    }
}
```

Example: Routing Initialization

```java
public class CreateNw implements Control {

    private int pid = 0;

    private static final String PAR_IDLENGTH = "idLength";

    private static final String PAR_PROT = "protocol";

    private static final String PAR_SUCCSIZE = "succListSize";

    int idLength = 0;

    int successorLsize = 0;

    int fingSize = 0;
    //campo x debug
    boolean verbose = false;

    public CreateNw(final String prefix) {
        pid = Configuration.getPid(prefix + "." + PAR_PROT);
        idLength = Configuration.getInt(prefix + "." + PAR_IDLENGTH);
        successorLsize = Configuration.getInt(prefix + "." + PAR_SUCCSIZE);
    }

    @Override
    public boolean execute() {
        for (int i = 0; i < Network.size(); i++) {
            final Node node = Network.get(i);
            final ChordProtocol cp = (ChordProtocol) node.getProtocol(pid);
            cp.m = idLength;
            cp.succLSize = successorLsize;
            cp.varSuccList = 0;
            cp.chordId = new BigInteger(idLength, CommonState.r);
            cp.fingerTable = new Node[idLength];
            cp.successorList = new Node[successorLsize];
        }
        final NodeComparator nc = new NodeComparator(pid);
        Network.sort(nc);
        createFingerTable();
        return false;
    }
```

- class CreateNw initialize:
  - successor lists
  - finger tables
- peer are ordered by ID simulating the chord ring

SHIFT + CTRL + H
type “CreateNw”
ENTER

Example: MessageCounterObserver

```
public boolean execute() {
    int size = Network.size();
    int totalStab = 0;
    int totFails = 0;
    ArrayList hopCounters = new ArrayList(); // struttura dati che
                                             // memorizza gli hop di
                                             // tutti i mess mandati
    hopCounters.clear();
    int max = 0;
    int min = Integer.MAX_VALUE;
    for (int i = 0; i < size; i++) {
        ChordProtocol cp = (ChordProtocol) Network.get(i).getProtocol(pid);
        // trovare tutti gli hopCOunter dei messaggi lookup mandati
        int[] counters = new int[cp.getLookupMessage().length];
        System.arraycopy(cp.getLookupMessage(), 0, counters, 0, cp
                .getLookupMessage().length);
        totalStab = totalStab + cp.stabilizations;
        totFails = totFails + cp.fails;
        cp.stabilizations = 0;
        cp.fails = 0;
        int maxNew = maxArray(counters, cp.index);
        if (maxNew > max)
            max = maxNew;
        if (cp.index != 0) {
            for (int j = 0; j < cp.index; j++)
                hopCounters.add(counters[j]);
            int minNew = minArray(counters, cp.index);
            if (minNew < min)
                min = minNew;
        }
        cp.emptyLookupMessage();
    }
    double media = meanCalculator(hopCounters);
    if (media > 0)
        System.out.println("Mean:  " + media + " Max Value: " + max
                + " Min Value: " + min + " # Observations: "
                + hopCounters.size());
    System.out.println("     # Stabilizations: " + totalStab + " # Failures: "
            + totFails);
    System.out.println("");
    return false;
}
```

SHIFT + CTRL + H
type
“MessageCounterObserver”
ENTER

Chord Configuration

```
# PEERSIM CHORD

# random.seed 1234567890
simulation.endtime 10^6
simulation.logtime 10^6

simulation.experiments 1

network.size 5000

protocol.tr UniformRandomTransport
protocol.tr.mindelay 0
protocol.tr.maxdelay 0

protocol.my ChordProtocol
protocol.my.transport tr

control.traffic peersim.chord.TrafficGenerator
control.traffic.protocol my
control.traffic.step 100

init.create CreateNw
init.create.protocol my
init.create.idLength 128
init.create.succListSize 12

control.observer MessageCounterObserver
control.observer.protocol my
control.observer.step 90000

control.dnet DynamicNetwork
{
    add 20
    add -25
    minsize 3000
    maxsize 7000
    step 100000
    init.0 ChordInitializer
    {
        protocol my
    }
}

```

- ***transport*** is the protocol to dispatch messages
- ***dynamicNetwork*** add / remove nodes from the network

Proposed exercises

- run Flooding
- change Flooding configuration to check how many peers are contacted modifying max step variable
- run Chord
- verify the differences with static and dynamic network
- modify Flooding to run K floods one after the other