Progress in the Development of the Distributed Simulation Tool P2PNetSim for analysis of large-scale P2P Networks

Miguel Rojas, Herwig Unger

ABSTRACT. In this paper, we present the progress of P2PNetSim development. P2PNetSim was started as a simulation tool initiative for large p2p networks by Coltzau [Col06]. We are aiming to add flexibility to support the needs of current P2P research, regarding to simulation architecture, usability and scalability. Since the lacking of a common existing tool for those kinds of simulations, we propose P2PNetSim as a powerful simulation environment for P2P Networks.

We hope this paper gives enough information in order to adopt this tool both in existing researching fields and new exploration areas. The simulator explained in this paper is flexible enough to support hundreds of thousands peers implementing different protocols, run defined test cases, and to gather statistics from properties of the simulated network elements, so it helps improving new P2P mechanisms or verifying and validating existing ones. Also we describe the architecture and show local and distributed executions of P2PNetSim by means of a use case.

RÉSUMÉ. A définir par la commande \resume{...}

KEYWORDS: P2P Network, Simulation Tool, Distributed System

MOTS-CLÉS:
1. State of the Art

This section gives an overview of some existing P2P simulation frameworks and compares them to our work. The SimP2 simulator [KI03] is designed to provide support and additional depth to an analysis of ad-hoc P2P-networks. The analysis is based on a non-uniform random graph model similar to Gnutella, and is limited to study basic properties such as reach-ability and nodal degree. 3LS [TD03] is a discrete simulator using a central step-clock. It provides three levels: Network level (bottom), Protocol model (middle) and User model (top). The network model uses a two dimensional matrix to define distance values between the nodes. The protocol model represents the P2P-protocol which should be investigated. Input can be simulated using the user model. Since 3LS is not efficient regarding memory usage, it is limited to rather small networks [ST04].

Narses Simulator [WS02] is a discrete-event and flow-based network simulator and thus does not concentrate on the packet-level to avoid the overhead of packet level simulators. To do this Narses offers a range of models that trade between fast runtimes and accuracy. Narses is therefore somewhere between packet level simulators and analytical models. Nevertheless the assumptions made by Narses are targeted towards reducing the complexity if simulations by approximations of physical aspects. Narses uses aiSee to visualization of P2P networks [aiS].

OMNeT++ is a discrete event simulation environment. Its primary application area is the simulation of communication networks, but other applications can be mentioned like the simulation of complex IT systems, queueing networks or hardware architectures as well, due to its generic and flexible architecture. Despite of OMNET++ was written in C++ there are version for multiple platforms such as Windows, Unix-like systems and Mac. There is an extension module named JSimpleModule that makes it possible to write OMNeT++ simple
modules in Java. Java and C++-based simple modules can be freely mixed in simulations. Integration is not seamless though, there are limitations like availability of OMNET features, as well special coding rules to obey in the Java code. The peer-to-peer simulator contains several models for structured (e.g. Chord, Pastry) and unstructured (e.g. GIA) P2P protocols.

PlanetSim [U.] is a Peer-to-Peer discrete-event simulator. It is written entirely in Java. The simulator supports both structured and unstructured overlays, but only implementations of two structured overlays, Chord-SIGCOMM and Symphony [MBR03] are distributed with the simulator. Although it is not possible to collect statistics from simulation runs, the simulator supports churn and node failure, has a visualiser that makes use of Pajek/GML, and simulations can be saved to disk for reuse. As we have outlined above, there are several java simulation environments. We support also to a pure java environment -due to their advantages and significant suitability for simulation [Jadb], [Gar01] - aiming for a powerful simulation tool that support the needs of current P2P research.

2. P2PNETSIM Simulation Tool

In the first release of P2PNetSim, the communication between components was based on the JADE system [Jada]. JADE is an agent-oriented communication environment which runs on the Java platform. It provides simple inter-agent communication mechanisms that are transparent for the user, i.e., developers do not have to care for the location of single agents. Despite of several advantages of JADE environment, in the new version of P2PNetSim this platform has been replaced by an own implementation. As will be mentioned later, this new architecture brings more flexibility in simulation time and other configurable properties.

The new implementation maintains the event-oriented architecture, where events are messages generated by sim-
ulated peers and sometimes by simulator agents. In each simulation step each peer executes its task and returns an event that will be collected by the local simulation agent. Simulation Agents will send the collected events to the Simulation Control Agent. This Agent is able to process all events before the next simulation step occurs.

![P2PNetSim Architecture](image)

Figure 1: P2PNetSim Architecture

A simulation will be terminated if the Controller detects a 'reset' event. Otherwise, the simulation continues in indefinite way. In the interconnecting network, each peer has connection parameters like incoming and outgoing bandwidth (in number of messages per simulation step) etc. The simulation environment additionally provides configurable transport latency for every subnet of peers.

P2PNetSim provides a platform for discrete-event simulations supporting object oriented models to utilize and extend the architecture. By this the potential for direct reuse of model code is maximized, while the dependencies on a particular simulator kernel implementation are minimized. The primary design goal was to support high performance simulations and to make models efficient.

Several classes have been developed to map the real P2P-behavior into simulation mode. Classes such as Peer, Message, Peer Event, EventLocalHandler and GlobalEventHandler en-
able the interaction between the Simulation Control Agent, Simulation Agent and the Simulated Peers. Events released by a Peer changes the status of the system. Message objects are used only for communication between Peers. For the purpose of gathering status information, the global event handler notify to the simulation system which are the events that might be logged and which not. Furthermore the events allow to entities, as peers as event handlers, change the status simulation. A simulation can be found in 'Running', 'Paused' or 'Reset' state.

Figure 2: Simulation State Diagram

2.1. Components

As a basic structure, the simulation consists of two different types of agents: Simulation Control Agent and Simulation Agent. Each agent has a unique identifier that can be used to address that agent transparently without knowing its location.

2.1.1. Simulation Control Agent (SCA)

The Simulation Control Agent’s responsibilities are:

– Running the simulation, i.e. synchronizing the Simulation agents, distributing the current time
and wait for the Simulation Agents to finish the current simulation step.

– Gathering status information from the peers via events, if necessary

– Control and Configuration of the simulation, i.e. adding, moving and deleting Simulation Agents and peers and change their configuration parameters

2.1.2. Simulation Agents (SA)

Several Simulation Agents are able to run the actual simulation, representing one IPsubnet each. They communicate with each other using RMI mechanisms. Simulation Agents provide (1) the communication layer to the simulated peers and are therefore able to route the peer’s messages to other peers that may run on other Simulation Agents and (2) the central point of events generated by peers located in a subnet.

2.1.3. Simulated Peers

Peers are represented by dynamically loadable modules that differ in their behavior as well as in their parameters (neighborhoods, network capabilities, etc.). From the implementation’s point of view, peers are provided by an interface to access the incoming and outgoing message queues, so that the simulation tool is able to deliver messages from one peer to another. Peers are created through the Simulation Control Agent’s API. Properties, such as address number, bandwidth, class name, etc, are initialized by the Simulation Agent.

2.1.4. Simulation’s Life Cycle

The Simulation Control Agent (SCA) is the central and main entity in charge of triggering the simulation. At starting point, this control agent set the peer’s parameters according to a configuration file. Once this initial phase is completed, the SCA invoke periodically 'doSim' method on each peer. After one step of execution is done over all peers, SCA gathers
events generated and saves their content into log files. If a 'pause' or 'reset' event is found, the SCA will stop the simulation process. In case of any 'reset' event the simulation will be re-initialized.

![Simulation State Diagram](image)

**Figure 3: Simulation State Diagram**

Additionally, after each simulation step and before the next one, the SCA delivers all messages sent between peers.

### 2.1.5. Simulation Configuration File

The configuration of the simulator is possible by using XML-files which define the topology, network elements (subnets and peers), event handlers and global properties (constants). Simulation agents can be configured for execution on different machines. In such a way, simulations are able to take advantage of several computers or cluster for running in distributed and parallel mode.
2.2. Application Programming Interface

Simulations are under supervision of SCA. The SCA, as mentioned above, is responsible of (1) synchronizing simulation agents in order to complete each simulation step, (2) gathering events released by peers and (3) delivering generated messages between peers. Within each one of these tasks, interfaces are required to allow the interaction between the simulation environment and the programmer’s classes.

2.2.1. Simulation Control Agent

After each simulation step, the SCA gathers events generated by the peers. From programmer’s viewpoint this task can be hooked via a global event handler. The global event handler, implemented by GlobalEventHandler class, has methods such as:

- `handlePeerEvents(SortedMap events, int time)`
  - Handle all events gathered from all simulator agents in the current simulation time

- `getSimulationPanel()`
  - Returns a panel that will be located in the simulation GUI

- `getConstant(String name)`
  - Returns the constant value associated to ‘name’. Constants are defined in the configuration file

- `getPeerProxies()`
  - Returns a set of PeerProxy. Each simulated peer has a proxiable object in SCA-side.

2.2.2. Simulation Agents

Simulations can be executed in local or distributed manner. On each machine can be configured and started a simulation agent. Those agents make possible (1) gathering peer events fired in each simulation step and (2) accessing to peers located in its sub-net. This behavior is offered by a local event handler and implements several methods such as:

- `handlePeerEvents(SortedMap events, int time)`
  - Handle all events gathered from all simulator agents in the current simulation time
- `getSimulatorName()`
  - Returns the simulator name

- `getConstant(String name)`
  - Returns the constant value associated to 'name'. Constants are initialized by SCA.

- `getPeers()`
  - Returns a list of peers which are under supervision of this agent.

The simulation environment includes default implementations for global and local event handler.

2.2.3. Simulated Peers

The most important interface from developer's viewpoint is the Peer class, since it is the super-class for executable peers under the simulation tool. This interface is modeled via the Peer class. The basic implementation of Peer class is responsible for (1) initializing the simulation parameters and constants sent by the SCA, (2) initializing the messages queues (input/output) and finally (3) executing a simulation step.

2.3. Integration with Apache ANT

P2PNetSim takes advantage of Ant build-files in order to run task such as (1) creating configuration files, (2) converting from spl to xml files and the most important (3) running a set of simulations sequentially.

3. USE CASE: UTH Overlay Networks & Compass Routing in UTH

3.1. Mesh-like Overlay Network

More and more daily goods are labeled with passive RFID chips and many manufacturers intend to use the huge amount of now newly available information to optimize their supply chain processes. However, mostly centralized network architectures and services, known as EPC global networks, are
used to transfer the RFID chip locations and information to the respective manufacturer.

In [BCS+07] intend to introduce an adaptive routing algorithm for passive RFID tag information, monitored from any internet-connected tag reader. This algorithm is based on building of mesh-like structures on top of a P2P-connected network of the manufacturer's servers. The platform consists of three levels: (1) Application level (responsible for the integration of the services), (2) Service level (responsible for representation of applications in the form of services, which can be contacted in a unified manner across a network) and (3) Middleware (responsible for communication between individual services). The developed UTH mesh-building algorithm works on the application level and is executed on every peer permanently. Each peer carries information such as logical node identifier (two equal sized dimensions X and Y derived from the EPC), its Internet Address and a vector of eight neighbors (one for each orientation: down, up, right, left, as well as upright, up-left, down-right and down-left).

The local node identifier builds a 2-dimensional space in which each node has an unique and well determined position through its identity. One possibility to link these nodes is to build up a mesh like structure. Therefore each node must find the closest neighbor in the cited 8 directions, what may be difficult since not all mesh cross-points necessarily have a corresponding active node. To build up the mesh structures in a self-organizing process, every peer receives requests to accept a node as a new neighbor. According to a fit-function these requests are accepted or rejected and therefore its neighborhood.

Simulation result have shown that all peers can find the maximum number of available neighbors and therefore, routings to all other neighbors can be established. The number of simulation steps needed to build the mesh increases, as expected, with the number of peers in the system. Still, it does
not even grow linearly, so $O(n)$ is definitely an upper border for this relation.

Additionally, figure 5 shows that the size of the mesh represented by the number of bits used in each dimension seems to only have a minor influence on the mesh building time.

### 3.2. Compass Routing

This section presents a routing algorithm which takes into account the above mentioned mesh-like structure. This algorithm, named Compass, intends on each routing step to use the best direction to forward a message. Because in such structures the best direction is not always present, the algorithm highlights a range of possible directions around of $(\pm) 90$ degrees.
To deliver a message from a source node to a target node, every peer must carry out the following functionality:

```plaintext
while ( exists ( message ) )
{
    target = extractTarget( message )
    best = getBestDirectionTo( target )
    if ( best != null )
        forward( message, best )
    return;
    set = getNearDirectionsOf( best, 90 )
    if ( isEmpty ( set ) )
        direction = selectDirectionRandomly();
    else
        direction = selectDirectionRandomFrom ( set );
    forward( message, direction )
}
```

Figure 5: Mesh building time
The selection of the next node in the message’s forwarding routine is done according to following:

- Forward the message using best direction if exist any neighbor pointing to that direction

- Select a neighbor randomly from those 90-degree adjacent directions.

Figure 6: Best and alternative directions in Compass Strategy

Here, the selection of the best direction in a 2-dimensional space is calculated by the Euclidean distance and angle between two points.

3.3. Simulation Results

Figure 7 shows the results for routing 10000 messages using compass strategy. It can be seen that the maximum number of a message needs to reach its goal is lower than 9 steps. From simulation results there were not lost messages.

Comparison of performance of Compass with Flooding algorithm was realized. The number of messages sent was of
10000 with TTL=9 for broadcasting. In Figure 8 can be seen that there is an overloading of total messages spreaded. Additionally there are a losing of messages due to TTL.

Finally compass strategy can be used as an alternative of flooding for message routing in UTH networks.

4. Conclusion & Future Work

Current approaches don’t support the simulation of required schema-based P2P networks. We have collected a set of requirements for the simulation of such networks and im-
implemented a corresponding simulation environment. We could confirm that P2PNetSim brings suitable performance by simulating of hundred of thousands peers. We are implementing routines to automatically change the size of networks by adding and removing nodes, named commonly as churn. Topologies such as a hypercube, random graph, regular rooted tree, scale free networks, a sink star, watts-Strogatz model should be implemented and integrated in tool. A transport layer that reliably delivers messages with a random delay that is drawn from the configured interval according to the uniform distribution might be also taking into account. We also plan to extend the simulator to support different response probability distributions, to model distinct amounts of content at peers.

References


