P2PAM: a Framework for Peer-to-Peer Architectural Modeling based on PeerSim

Matteo Agosti
Distributed Systems Group
Information Technology Department
University of Parma (Italy)
agosti@ce.unipr.it

Michele Amoretti
Distributed Systems Group
Information Technology Department
University of Parma (Italy)
amoretti@ce.unipr.it

Francesco Zanichelli
Distributed Systems Group
Information Technology Department
University of Parma (Italy)
zanichelli@ce.unipr.it

Gianni Conte
Distributed Systems Group
Information Technology Department
University of Parma (Italy)
conte@ce.unipr.it

ABSTRACT

A peer-to-peer architectural model defines an overlay network topology and a routing strategy. If these aspects are tied together by a deterministic logical model, we say that the architecture is structured. Otherwise, we say it is unstructured. Based on these assumptions, in recent years many complex P2P architectural models have been defined, their performance evaluation being carried out mainly by means of simulative tools. However, there is an emerging need for a general-purpose tool, enabling large-scale overlay network simulations, yet also providing ready-to-use complex building blocks. The widely known PeerSim simulator addresses the first issue quite effectively, although it appears quite limited with respect to several important aspects, *i.e.* churn modeling. In this paper we propose P2PAM as a PeerSim enhancement providing a rather complete framework for peer-to-peer architectural modeling. P2PAM effectiveness is demonstrated by showing how it has been used to rapidly develop simulations of two interesting systems, namely JXTA and HALO.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Performance of Systems**]

1. INTRODUCTION

Successful deployment of extensive Virtual Organizations calls for efficient workload distribution and high resource availability. These goals cannot be fully guaranteed by the client/server approach, due to the fact that user-owned resources may remain unused. Furthermore, shared resources are usually published/searched through a centralized repository or broker, thus introducing single points of failure and possibly yielding to scalability issues. It is also for these reasons that peer-to-peer interaction has emerged as a promising new paradigm for distributed computing, aiming at efficient workload distribution and high resource availability [2]. The main idea behind the peer-to-peer paradigm is that each peer, *i.e.* each participant, can act both as a client and as a server *in the context of some application.*

A peer-to-peer architectural model defines an overlay network topology and a routing strategy. If these aspects are correlated by a deterministic logical model, we say that the architecture is *structured.* This is the case, for example, of a peer-to-peer system in which nodes and message identifiers are taken from the same space, the overlay network topology is a tree, and propagation is based on choosing the neighbor whose identifier is mostly similar to the message identifier. On the other hand, if there is no underlying deterministic logical model, we say that the architecture is *unstructured.* Moreover, topologies can be classified as centralized, partially centralized (hybrid), and decentralized (pure), according to the taxonomy of physical networks proposed by Paul Baran in the early sixties [6].

In most cases, it is overly difficult to characterize the performance of a peer-to-peer architectural model by means of analytical tools. On the contrary, simulation studies allow the characterization of the large number of parameters which typically define the behaviour of a peer-to-peer network. However, there is an emerging need for a general-purpose tool, enabling large-scale overlay network simulations, yet also providing ready-to-use complex building blocks. The PeerSim simulator [11], written in Java, addresses the first issue quite effectively, but it appears quite limited with respect to several important aspects, *i.e.* churn modeling. In this paper we propose P2PAM [1] as a PeerSim enhancement,

Simutools 2008 March 3-7, 2008, Marseille, France.

[1] http://dsg.ce.unipr.it/research/P2PAM/p2pam.html

providing an almost complete framework for peer-to-peer architectural modeling.

The effectiveness of P2PAM is demonstrated by showing how it has been used to rapidly develop simulations of two comparable routing protocols: JXTA-SRDI, which is the default solution in JXTA [18], and HALO, a solution we designed in order to exploit the peculiarities of scale-free network topologies.

The paper is organized as follows. Related work on peer-to-peer simulators is discussed in section 2. The P2PAM framework is described in section 3, starting from its general organization and basic class structure to advanced features such as network dynamics management. Section 4 illustrates the P2PAM-based simulation of JXTA-SRDI and HALO, with different overlay network topologies. Finally, an outline of open issues concludes the paper.

2. RELATED WORK ON P2P SIMULATORS

The simulative approach is becoming the most common technique to study overlay networks and P2P applications. The cost of implementing a solution into a simulation environment is considerably lower than what is required to realize a similar experiment on geographical networks. Specifically, the number of computational resources needed is lower and the simulated model can be built to be more realistic than any other tractable mathematical model. The use of a simulation environment may enable the detailed evaluation of architectural models and allow for an high reuse of code when the devised solution will be experimented in the real world.

We analyzed different simulators and realized that no commonly agreed reference architecture exists yet; additionally, only few P2P-related papers report about the simulation environment used to obtain the presented results. The absence of standards leads to the lack of common analysis instruments and makes impossible to reproduce and verify results with different simulators than those used to obtain the original results.

In order to choose the proper simulation environment to be used as starting point for the development of P2PAM, we evaluated different systems according to a set of criteria similar to those presented in [14]:

- *Simulation Architecture*: the operation and the design of the simulator.
- *Usability*: how easy the simulator is to learn and use.
- *Extensibility*: the possibility to modify the standard behavior of the simulator in order to support specific protocols.
- *Configurability*: how easily the simulator can be configured and with which level of detail.
- *Scalability*: the ability to simulate how a P2P protocol scales with thousands, or more, nodes.
- *Statistics*: how much the results are expressive and easy to manipulate.
- *Reusability*: the possibility to use the simulation code to write the real application.

The most used system for simulating application level protocols is *NS-2 Network Simulator* [19], even if it was originally designed to work at network level. NS-2 is written in C++ and uses the object-oriented paradigm. It offers a discrete-event model and an OTcl [17] interpreter as a front-end. However, as NS-2 models both physical and link substrates with high level of detail, it is not very scalable, that is the maximum network size amounts to $4 \cdot 10^3$ nodes.

P2PSim [10] is a discrete-event simulator for structured overlay networks written in C++. P2PSim supports several peer-to-peer protocols including the recent Koorde and Kademlia, however the different underlaying network models are implemented with a rather abstract level of detail. The lack of documentation makes it hard to extend P2PSim, whereas its scalability is limited to a maximum of $3 \cdot 10^3$ nodes.

OverlayWeaver [15] is a peer-to-peer overlay construction toolkit written in Java, that provides a common API for higher-level services and a set of routing algorithms like Chord, Kademlia and Koorde. The toolkit contains a so-called Distributed Environment Emulator which invokes and hosts multiple instances of Java applications on a single computer; due to the threads limits imposed by the Java Virtual Machine, the scalability is limited to $4 \cdot 10^3$ nodes. Unfortunately the emulator does not provide network statistics, thus limiting its utilisation as a simulator.

PlanetSim [20] is a discrete-event simulator developed in Java, that offers a layered and modular architecture. Distributed services in the simulator uses the Common API for structured overlays enabling the reusability of simulation code to experimentation code running in the Internet. As for OverlayWeaver, it is not possible to collect statistics from the simulation outputs. PlanetSim offers a network layer wrapper which allows to port the simulation code to real networks like PlanetLab; however, this partial support for network protocols limits the scalability, making PlanetSim able to simulate networks with size in the order of 10^5 nodes.

PeerSim [11] enables the simulation of structured and unstructured networks by using either a cyclic model or a discrete-event mode. It is completely written in Java and offers a well documented lightweight API that makes it easy to modify the standard behavior of the simulator. PeerSim enables the implementation of personalized components, so-called observers, in order to export custom statistical indicators on the simulation results. PeerSim offers the best scalability among analyzed simulators as it can reach up to 10^6 nodes by using the cyclic model. The event model is less efficient than the cyclic one, but is more realistic because it enables the simulation of protocol stacks. PeerSim can be configured by means of a plain text file, defining scheduling and parameter values for each component. Developers can easily access the configuration manager in order to make more customizations.

3. P2PAM

The PeerSim architecture was designed in order to provide a simple API to develop application level P2P protocols and simulate their execution. The result is a set of simple Java classes with the purpose of showing how to use the static objects that represents network nodes rather than give a starting point for the development of extensions. PeerSim comes with few simple topologies and network observers, but it lacks any routing protocol. Moreover, the concept of node is extremely simplified and the concept of *resource* is never specified.

The real advantage of PeerSim is the engine that supports many estensible and pluggable components, with a flexible configuration mechanism, but the researcher that wants to evaluate its own protocols has to build everything from the scratch. Since the community of P2P researchers is large, this can result in many non-interoperable or differently optimized packages (the NS-2 experience is emblematic, in this sense).

The basic idea of P2PAM is to add another abstraction layer to PeerSim in order to let the developer focus only on routing protocol implementation and performence analysis. P2PAM comes with a set of network topologies defined across a protocol-level interface which simplifies the task of routing protocol development. The concept of node has been extended in order to make it more similar to a real peer node, with estensible data structures storing local resource descriptions, search queries, etc. A simulation automator has been introduced in order to simplify the task of simulating network dynamics (churn, publication and search) using a simple scripting language, rather than writing Java code and re-compiling each time.

3.1 Organization of the Framework

P2PAM has been divided into packages, each one addressing a specific aspect of the simulation. The root package is org.dsg.p2pam and contains the following sub-packages:

- automator including the Simulation Automator's classes;
- init including the default node initializer;
- node including node data structure classes;
- observer including network and resource observers;
- routing including the routing protocols reference implementation;
- topology including network topology classes;
- util including classes for key generation, resource distribution and configuration file management.

Each package provides a set of basic interfaces and reference implementations.

3.2 Node Data Structure

The node package contains node-related data structures and functionalities. In particular, the NodeDataCollector class, which implements PeerSim's Protocol interface, provides get and set methods for managing the basic features of each peer:

- type (leaf or supernode)
- unique identifier
- availability (is the node public?)
- list of local resources
- list of cached queries

Clearly, the NodeDataCollector class can be extended in order to add architecture-specific features.

3.3 Node Initializer

The init package provides the necessary tools to initialize network nodes and distribute resources according to a specific strategy (*i.e.* Zipf, uniform, etc.). The NodeInizializer class processes the network node-by-node, doing the following:

- determine if the node is public, according to a maximum number of public nodes;
- determine if the node is supernode or leaf;
- assign to the node a set of resources whose size is evenly distributed between a minimum and maximum amount.

The role of a node initializer is to allow the full setup of each node, except link establishement which is performed by topology constructors.

3.4 Topology Constructors

The topology package provides the TopologyBuilder abstract class, which extends PeerSim's WireGraph class. The TopologyBuilder declares three abstract methods, respectively wire(), addNode(), and removeNode(), which must be implemented by every subclass.

In the following we describe two important overlay topology models which are provided by P2PAM, as implementations of the TopologyBuilder abstract class. To describe the related network topologies we consider the distribution of their node degree, adopting the *probability mass function (PMF)* formalism, which comes from the theory of random graphs [7]:

$$P(k) = P\{\text{node degree} = k\}$$

3.4.1 Poisson Random Networks

The first and most investigated random graph model has been introduced by Erdös and Rényi (*ER model*). Networks based on the ER model have N nodes, each one connected to an average of $\alpha = \langle k \rangle$ nodes. The presence or absence of a link between two nodes is independent of the presence or absence of any other link, thus each link can be considered to be present with independent probability p. For large N, the degree distribution of the network converges to the Poisson distribution

$$P(k) = \frac{\alpha^k e^{-\alpha}}{k!} \qquad \text{with } \alpha = \langle k \rangle = \sigma^2 \tag{1}$$

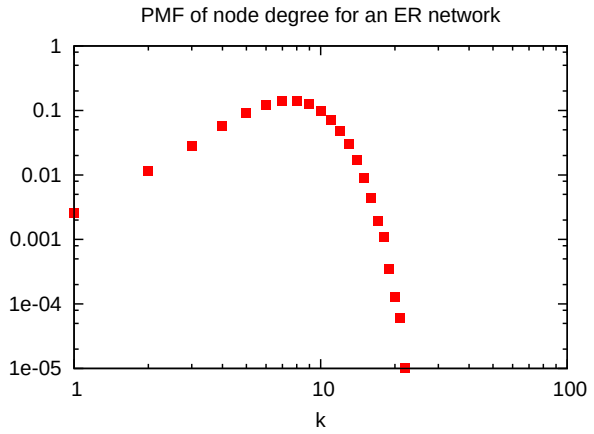

Figure 1: Node degree distribution for a simulated ($N = 10^5$ nodes) Poisson random network, with average node degree $\alpha = 7$ (in log scales).

An example of Poisson distribution of the node degree is illustrated in figure 1.

The ERTopologyBuilder class implements the ER model, and it is provided in topology package along with its base abstract class TopologyBuilder.

3.4.2 *Scale-free Networks*

Poisson networks are fairly homogeneous, *i.e.* each node has approximatively the same number of links. In contrast, studies about the World Wide Web, the Internet, and other large networks indicate that these systems belong to a class of inhomogeneous networks, for which the node degree PMF decays as *power law* [16, 13] distribution:

$$P(k) = ck^{-\tau} \tag{2}$$

with $\tau > 1$ (to be normalizable), and

$$c = [\sum_{k=1}^{\infty} k^{-\tau}]^{-1} = [\zeta(\tau)]^{-1} \tag{3}$$

where $\zeta(\cdot)$ is the Riemann zeta function. These networks are called *scale-free*, because their separation degree growth is sublinear with respect to N. In particular, if $2 < \tau < 3$, the diameter is $d \sim \ln \ln N$ [8]. Even if the number of nodes strongly increases, the mean distance between two nodes remains the same.

Barabási and Albert proposed a simple model (called *BA model*) [4, 5, 1] to construct scale-free networks with $\tau \simeq 3$. The BA model is based on two ingredients: growth (*i.e.* N should not be fixed in advance), and preferential attachment (*i.e.* the probability with which a new node connects to the existing nodes is not uniform as in Poisson random networks). P2PAM provides a BA model implementation, for which each new node joining the network connects to m existing nodes of the system, with probability $\Pi(k, N)$ that the $(N+1)$-th node will be connected to any node with degree k being dependent on the node degree k of that node.

The most elegant way of deriving the node degree distribution of a network constructed with the BA model, is the *master-equation approach* proposed by Dorogovtsev and Mendes [9]. The resulting PMF is

$$P(k) = \frac{2m(m+1)}{k(k+1)(k+2)} \simeq 2m^2k^{-3} \quad \forall k \geq m \tag{4}$$

which is a power law with exponent $\tau = 3$.

Figure 2 illustrates the node degree distribution of network based on the BA model.

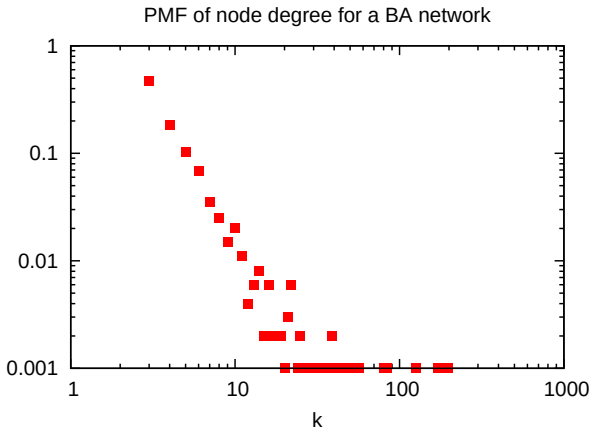

Figure 2: Node degree distribution for a simulated ($N = 10^5$ nodes) BA network, with $m = 3$ and $N_0 = 5$.

The BATopologyBuilder class implements the BA model, and it is provided in topology package along with its base abstract class TopologyBuilder.

3.5 Resource Replication and Popularity

By replication, we mean the number of nodes that have a particular resource. Popularity determines the frequency of queries for individual resources. We assume that there are m resources of interest, and q_i represents the relative popularity, in terms of number of queries issued for it, of the ith resource. Values are normalized, *i.e.*

$$\sum_{i=1}^{m} q_i = 1$$

Examples of popularity are:

- Uniform: $q_i = 1/m$
- Zipf-like: $q_i \propto 1/i^{\tau}$ with τ close to 1

We assume that each resource is replicated on r_i nodes, and the total number of resources in the network is M.

$$\sum_{i=1}^{m} r_i = M$$

Examples of replication distributions are:

- Uniform: $r_i = M/m$
- Proportional: $r_i \propto q_i$

- Square-root: $r_i \propto \sqrt{q_i}$
- Zipf-like: $r_i \propto 1/i^{\tau}$ with τ close to 1

The square-root replication minimizes the overall search traffic [12], and can be obtained with the *path replication* strategy: when a search succeeds, the resource is stored at all nodes along the path from the requester node to the provider node. This is quite straightforward if the resource is a file, but difficult for other types of resources (*e.g.* services).

P2PAM allows to choose the resource distribution, but also to define, for each node, the minimum and maximum number of owned resources.

3.6 Network Dynamics

3.6.1 Churn

Node arrival and departure (*churn*) has many effects, in a P2P system: data unavailability, routing table inconsistency, overlay network fragmentation. Adequate strategies must be implemented in order to guarantee the system's adaptivity to any churn rate, resulting in a constant performance degree of all its functionalities.

P2PAM allows to simulate the dynamics of a network topology by allowing to introduce node departures, which can be combined with previously described network growth models. In this section we illustrate the parameters which are involved in this process.

Peer departures begin after nodes_before_departures nodes have joined the network. At this point, for every new node joining the network, a random number in $[0, 1]$ is generated and compared with departure_probability. If higher, there are node departures. The number of leaving nodes is computed as randomly generated percentage of the number of connected nodes. Three parameters allows to characterize this phase: min_departures_percentage, max_departures_percentage and max_departures, the latter being an absolute limit to the number of departures for step.

Leaving nodes can be randomly chosen, or not. For example, in a scale-free network, it could be assumed that highly connected nodes are stable. In this case the possible strategy for deciding if a randomly chosen node must leave the network is to compare its node degree with the average one: if higher, the node does not leave the network. To add more realism, the hub_departure_probability parameter can be used to set a leaving probability also for highly connected nodes. The same parameters can be used for many different strategies.

It is also possible to define the fraction of nodes which notify (*e.g.* to their neighbors) their intention to leave the network. Otherwise, the departure is considered as node failure, which will be managed asynchronously by the network.

It is also possible to configure an oscillatory behavior, in which the network size alternately decreases from and increases to an established size value. The number of ripples is defined by the waves parameter. Each ripple is characterized by a number of departures comprised between the wave_min_departures and wave_max_departures values. The oscillation ends once the number of departures has been compensated by the number of arrivals. Of course, all parameters can be set in order to achieve particular behaviors, such as the complete wipeout of the network.

3.6.2 Publication and Search

The routing package includes the RoutingProtocol abstract class, which declares two abstract methods: publish() and search(). The concretization of the publish() method must provide routing mechanisms for spreading resource descriptors (*e.g.* key identifiers) all over the network, according to a specific strategy (*e.g.* Gnutella, Chord, etc.). Similarly, the search() method must realize a resource discovery process (keyword-based, ontology-based, etc.). Two implementations of the RoutingProtocol abstract class are discussed in section 4.

3.7 Simulation Automator

Oftentimes the evaluation of a P2P routing protocol involves the simulation of different behaviors in terms of network churn, search and publication of services. In order to do this, the developer must continuously modify the source code and recompile everything. To simplify this process, P2PAM provides a SimulationAutomator, which is a parser of a basic scripting language that enables the configuration of the simulations without the need to recompile the source code. At this very early stage of development, the automator, which is a simple PeerSim control, takes as unique parameter a script file that defines the network dynamics through a list of <action, amount> pairs. The action field can assume the following values:

- add - simulates the entrance of amount new nodes into the network;
- fail - simulates the failure of amount node into the network;
- remove - simulates the clean disconnection of amount nodes of the network;
- publish - simulates the publication of amount resources, randomly chosen from those available in node caches;
- search - simulates amount search queries.

A typical use case is to grow a network until it reaches a fixed size, and then start executing different kinds of action. The simulation stops when all actions have been completed, according to their amount value.

Despite the simplicity of this syntax we were able to simulate different network dynamic behaviors without recompiling our code. We have planned to extend the SimulationAutomator in order to use XML as scripting language, and to support the simulation of complex network dynamics, such as concatenations of different evolution phases.

3.8 Observers

To record data during a simulation and to compute performance indices, P2PAM provides specific classes, called

observers, which are extensions of PeerSim's GraphObserver class. Observers allows to analyze:

- the overlay network topology (distribution of the node degree, average connected distance, clustering coefficient);
- the distribution of resources and resource advertisements;
- the data structure of each node (*e.g.* its knowledge base);
- the results of the search processes (query hit ratio, precision and recall);

Since the network topology changes dynamically during the publication and search processes, it is possible to define multiple observation instants, at which different *shots* are taken.

Observed connections can be stored in a file for further reuse (with PeerSim's WireFromFile class). This feature is useful to quickly debug routing protocols, avoiding to repeat the construction of the topology.

4. EXPERIMENTAL EVALUATION

This section illustrates the simulation of two interesting routing strategies, *i.e.* JXTA-SRDI and HALO, with different unstructured network topologies. The two protocols have been implemented in P2PAM respectively in the JxtaRoutingProtocol and HaloRoutingProtocol classes, both based on the RoutingProtocol abstract class described in section 3.

4.1 JXTA Routing

Project JXTA [18], originally conceived by Sun Microsystems, and designed with the participation of a growing number of experts from academic institutions and industry, defines a generic peer-to-peer network overlay usable to implement a wide variety of applications and services. The JXTA platform provides core building blocks (IDs, advertisements, peergroups, pipes) and a default set of core policies, which can be replaced if necessary.

JXTA-SRDI is the default protocol for message routing for resource sharing and discovery in JXTA networks. It is based on two components: the *Shared Resource Distributed Index (SRDI)*, and the *loosely-consistent DHT walker*. The SRDI module implemented in each JXTA peer is used on one hand to extract entries from resource advertisements and push them to the network, and on the other hand for lookups. The walker is used for routing when no index information is locally available.

JXTA supernodes are called rendezvous super-peers, while leaf nodes are called edge peers. Resources, services, peers and peergroups are described by XML documents, the so-called advertisements. When an advertisement is published by an edge peer (E1), its entries (which are attribute-value pairs) are sent to the connected supernode (R1). Supernodes store the entries in dynamic indexes including also, for each entry, the ID of the peer which originated them and an expiration time. Moreover, each index entry is replicated from supernode R1 to another supernode in R1's RPV using a DHT function. In details, the 160 bit SHA1 hash address space is evenly divided amongst the ordered RPV (sorted by peer ID), so an index entry is routed by hashing its value and mapping its location in the RPV (suppose R11). If the RPV is > 3, each index entry is also replicated to the RPV neighbors of R11 ($+1$ and -1 in the RPV ordered circular list).

Search is based on the same DHT function, but also on a limited range walker to resolve inconsistency of the DHT within the dynamic rendezvous network. Queries are messages which contain advertisement entries (attribute-value pairs). If a query is sent by an edge peer (E2), it reaches the connected supernode (R2). Once R2 receives the query, it first attempts to match it locally. Then R2 forwards the query to another supernode in R2's RPV (suppose R22) using the DHT function. The more the rendezvous network is near to completeness, *i.e.* the RPV is consistent across all supernodes, the more the DHT-based routing algorithm is efficient. To compensate for any RPV skew, a *limited range walker* is used. For example, suppose R22 fails to match the query, and its RPV is

R20 R21 *R22* R23

Assume that R20 and R21 have the same RPV of R22, while R23 has the following RPV:

R23 R24 R25

Thus an R22 originated limited range walker query is walked to R21 with a TTL (time to live, *i.e.* maximum number of hops) of 2, and to R23 with a TTL of 1, where the TTL is adjusted to 2 on R23 and walked to R25. When there is a query hit, the response is forwarded to the query originator (in this case, edge E2).

The more the supernode network is near to completeness, *i.e.* the peerview is consistent across all supernodes, the more the routing algorithm is efficient. The strategy is summarized in algorithm 1.

4.2 HALO Routing

Our HALO protocol is based on high-degree node search, for which messages are routed choosing at each step the highest-degree neighbor, and using the DHT function for corrections and for the final hop. The idea for this strategy comes from the observation that the pseudo-DHT strategy adopted by JXTA gives its best when the supernode network is highly connected. HALO routes messages towards best connected nodes, and uses the same DHT function used by JXTA if neighbors are less connected than current peer, and for the final step. If many neighbors have the same highest node degree, the target is chosen by proximity of its ID with the message ID. Algorithm 2 summarizes the HALO strategy. Note that search and publishing strategies are exactly symmetrical. The maximum number of hops for a message is TTL.

```
1: if (publication) then
2:    save locally
3: end if
4: if (search) then
5:    match locally
6: end if
7: if (leaf peer) then
8:    send message to supernode
9: end if
10: if (supernode) then
11:    if ((message from leaf peer) || (local message)) then
12:       find target supernode neighbor t using DHT function
13:       send message to target t supernode neighbor
14:       if ((publication) && (peerview > 3)) then
15:          send message to t+1 and t-1 supernode neighbors
16:       end if
17:    end if
18:    if (message from supernode) then
19:       if ((search) && (no local match)) then
20:          walk the peerview
21:       end if
22:    end if
23: end if
```

Algorithm 1: JXTA message routing.

```
1: if (publication) then
2:    save locally
3: end if
4: if (search) then
5:    match locally
6: end if
7: if (leaf peer) then
8:    send message to supernode
9: end if
10: if (supernode) then
11:    if (Hops < TTL - 1) then
12:       search for supernode neighbor with higher degree
13:       if (found n ≥ 1 supernode neighbors with higher degree) then
14:          choose highest degree supernode neighbor with ID ≃ message ID
15:          send to chosen supernode neighbor
16:       else
17:          find target supernode neighbor t using DHT function
18:          send message to target t supernode neighbor
19:       end if
20:    end if
21:    if (Hops == TTL - 1) then
22:       find target supernode neighbor t using DHT function
23:       send message to target t supernode neighbor
24:    end if
25: end if
```

Algorithm 2: HALO message routing.

4.3 Performance Comparison

One of the issues which arise when simulating a complex architectural model is the out-of-hand growth of memory requirements, in particular when the network size increases by factors of ten. The system we used to run the tests is a dual processor 2GHz AMD Opteron 246 (a 64bit machine) with 1024KB cache and 6GB RAM, equipped with Linux kernel 2.6.5-7.283, and Java Standard Edition version 1.5.0 10-b03. With this platform we were able to simulate 10^6 node networks, but since the focus of this work is to evaluate the flexibility of P2PAM as a tool for rapid application protocol development, we chose to limit the network size to 10^5 nodes in order to shorten the execution time of each configuration. We set the fraction of supernodes to 5% of the network size, since it is a realistic value for JXTA (as we illustrated in a previous work [3]).

We chose to operate only on a significant set of parameters, such as the *TTL of service discovery messages*, the *average node degree* α of the ER topology, and the *initial node degree* m of the BA topology. Other variables have been fixed to reasonable values, minimally affecting the computational complexity.

Each node is interested in a limited number of resources among the whole set of resources which are spread in the network with replication according to a Zipf-like distribution. We considered 10000 different resources, with 5 to 15 resources for each node. Obviously the probability of having a particular resource increases with the size of the network.

The query hit ratio (QHR) is the fraction of successful queries, *i.e.* queries with at least one useful response. Typically, QHR values over 90% denote high efficiency of the search algorithm, but also a good distribution of resource advertisements in the network.

4.3.1 JXTA-SRDI vs HALO Efficiency for Different TTL Values

Figures 3 and 4 illustrate, for JXTA-SRDI and HALO, how the QHR changes depending on the TTL, for three different sizes of the network, considering both ER and BA topologies.

The number of different resources is an important parameter. We observe that the QHR, for both routing algorithms, is significantly influenced by the network size. The Zipf-like resource distribution does not guarantee that all categories are present, unless the network size is greater than the ratio between the number of categories and the average number of peer resources. With 10^4 categories, 5 to 15 resources for each peer, it appears that 10^3 nodes are not enough to guarantee the presence of all resource categories, thus the QHR is always below 80%. It is sufficient to increase the network size by one order of magnitude, to be quite sure that searched resources are actually owned by at least one peer. In this circumstance, search results depend only on the efficiency of the routing algorithm.

Finally we observe that with ER topology JXTA-SRDI is slightly more efficient than HALO. Their performance indicators converge for high TTL values. With BA, under a

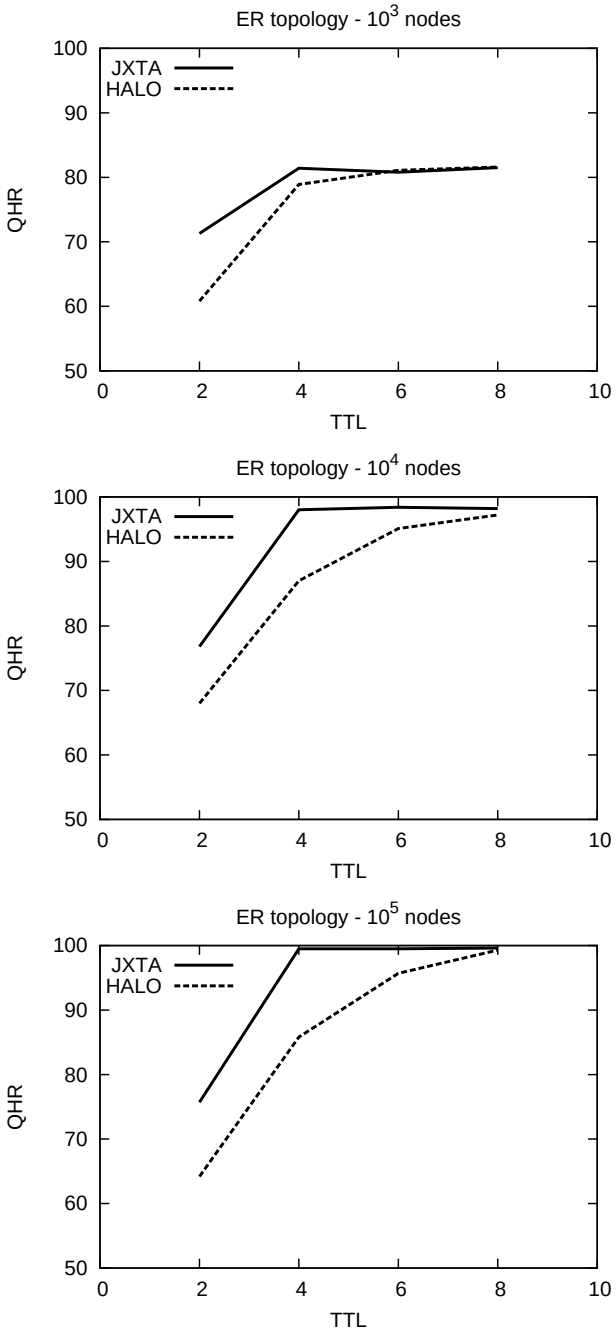

Figure 3: Query hit ratio versus TTL, for JXTA-SRDI and HALO strategies, with ER topology model. Three network sizes have been considered: respectively with 10^3, 10^4, and 10^5 nodes.

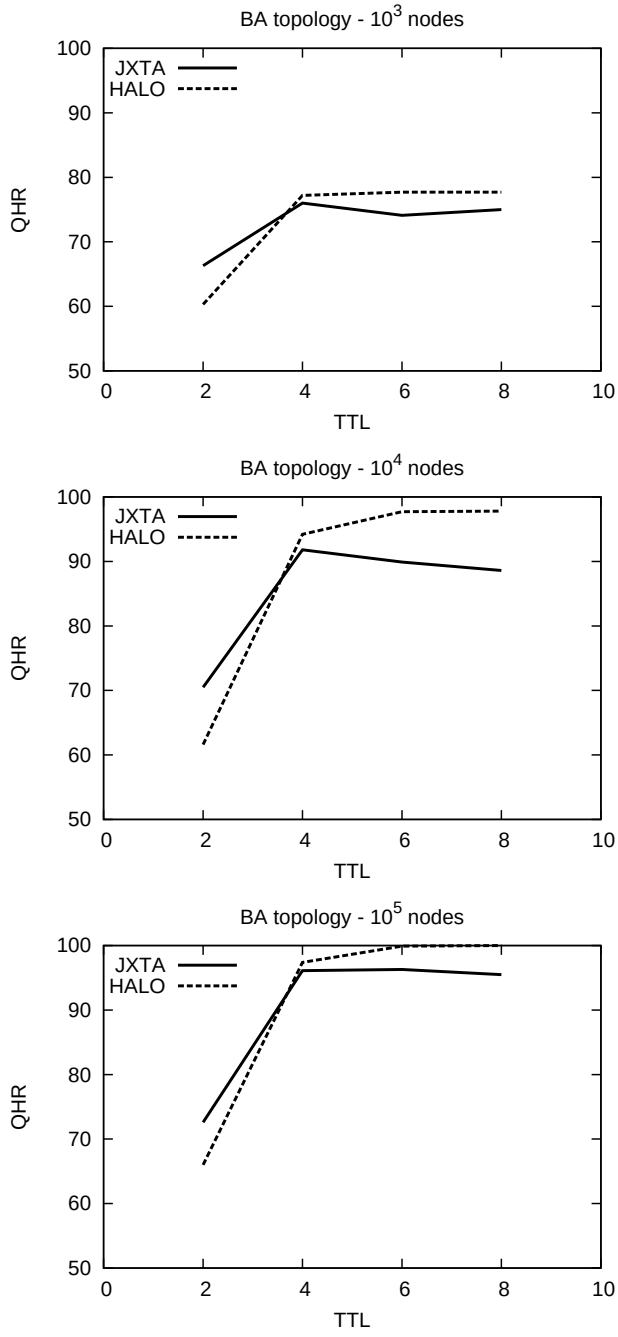

Figure 4: Query hit ratio versus TTL, for JXTA-SRDI and HALO strategies, with BA topology model. Three network sizes have been considered: respectively with 10^3, 10^4, and 10^5 nodes.

Table 1: JXTA efficiency in ER topologies for different α values.

α	Query Hit Ratio
3	90.3
5	97.5
7	99.5
9	99.9
11	96.3

Table 2: HALO efficiency in BA topologies for different m values.

m	Query Hit Ratio
3	99.9
5	100.0
7	100.0
9	100.0
11	100.0

TTL threshold JXTA-SRDI performs better than HALO, but over the threshold JXTA-SRDI perfomance is constant while HALO perfomance increases and overcomes JXTA-SRDI's one.

4.3.2 JXTA-SRDI and HALO Efficiency for Different Node Degree Values

From prevously illustrated results, it appears that JXTA-SRDI is more efficient than HALO on ER topologies, while HALO best performs on BA topologies. In this section we compare the two algorithms, each one running on the best suited topology.

Table 1 shows the efficiency of JXTA-SRDI with ER topologies, for increasing α values, with $R = 5000$ supernodes among $N = 10^5$ total nodes in the network, and $TTL = 6$. We observe that JXTA-SRDI efficiency is near to excellence for α values between 5 and 9.

Table 2 shows the efficiency of HALO with BA topologies, for increasing m values, with $R = 5000$ supernodes among $N = 10^5$ total nodes in the network, and $TTL = 6$. HALO efficiency is unaffected by m variations.

5. CONCLUSIONS

In this paper we illustrated a PeerSim enhancement, called P2PAM, providing an almost complete framework for peer-to-peer architectural modeling. P2PAM comes with a set of network topologies defined across a protocol-level interface which simplifies routing protocol development and simulation. The concept of node has been extended in order to make it more similar to a real peer node, with estensible data structures storing local resource descriptions, search queries, etc. A simulation automator has been introduced in order to simplify the task of simulating network dynamics (churn, publication and search) using a simple scripting language, rather than writing Java code and re-compiling each time.

The effectiveness of P2PAM is demonstrated by showing how it has been used to develop simulations of two comparable routing protocols, JXTA-SRDI and HALO, with different unstructured network topologies. For both protocols, we measured the query hit ratio considering different configurations of a significant set of parameters. Many other parameters could have been manipulated, without changing and re-compiling the Java code.

Due to its ease of use, P2PAM will be essential for our study of complex peer-to-peer architectural models, considering different resource distributions but also different degrees of churn rate. To this purpose, we plan to extend the simulation automator in order to use XML as scripting language, and to support the simulation of complex network dynamics, such as concatenations of different evolution phases. Another improvement will be the possibility of storing observed connections among nodes in a database, in order to ease the debugging of routing protocols, without reconstructing the topology each time.

Acknowledgements

This work has been partially supported by the Italian Ministry for University and Research (MIUR) within the project PROFILES under the PRIN 2006 research program.

6. REFERENCES

[1] R. Albert and A. Barabási. Statistical mechanics for complex networks. *Reviews of Modern Physics*, 74(1):47–97, January 2002.

[2] M. Amoretti, M. Reggiani, F. Zanichelli, and G. Conte. Peer: an Architectural Pattern Enabling Resource Sharing in Virtual Organizations. In *The 12th Pattern Languages of Programs (PLoP) 2005, Monticello, Illinois, USA*, September 2005.

[3] M. Amoretti, M. Reggiani, F. Zanichelli, and G. Conte. SP2A: Enabling Service-Oriented Grids using a Peer-to-Peer Approach. In *Workshop on Emerging Technologies for Next generation GRID, 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprise (WETICE-2005), Linkoping, Sweden*, June 2005.

[4] A. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, October 1999.

[5] A. Barabási, R. Albert, and H. Jeong. Mean-field theory for scale-free random networks. *Physica A*, 272(1-2):173–187, October 1999.

[6] P. Baran. Introduction to Distributed Communications Network. Technical report, RAND Corporation, Aug. 1964.

[7] B. Bollobás. *Random Graphs*. Academic Press, 1985.

[8] R. Cohen and S. Havlin. Scale-free networks are ultrasmall. *PHYS.REV.LETT*, 90:058701, 2003.

[9] S. Dorogovtsev and J. Mendes. Evolution of networks. *Advances in Physics*, 51:1079, 2002.

[10] T. M. Gil, F. Kaashoek, J. Li, R. Morris, and J. Stribling. p2psim: a simulator for peer-to-peer (p2p) protocols. http://pdos.csail.mit.edu/p2psim/.

[11] M. Jelasity, A. Montresor, G. Jesi, and S. Voulgaris. PeerSim: A Peer-to-Peer Simulator. http://peersim.sourceforge.net, 2004.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *16th International Conference on Supercomputing, New York*, June 2002.

[13] M. Mihail and C. Papadimitriou. On the Eigenvalue Power Law. In *6th International Workshop on Randomization and Approximation Techniques, Cambridge, Massachusetts*, September 2002.

[14] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *Computer Communication Review*, 37(2):95–98, 2007.

[15] K. Shudo, Y. Tanaka, and S. Sekiguchi. An overlay construction toolkit. http://overlayweaver.sourceforge.net.

[16] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos. Power-laws and the AS-level Internet Topology. *IEEE/ACM Transactions on Networking*, 11(4):514–524, August 2003.

[17] The Berkeley Multimedia Research Center. OTcl - object tcl extensions. http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/.

[18] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J. Hugly, E. Poyoul, and B. Yeager. Project JXTA 2.0 Super-Peer Virtual Network. Technical report, Sun Microsystems, May 2003.

[19] University of California Information Sciences Institute. NS-2 network simulator. http://nsnam.isi.edu/nsnam/index.php.

[20] University Rovira i Virgili. Planetsim project. http://planet.urv.es/trac/planetsim.