JXTA-Sim
A simulator for evaluating the JXTA Lookup Algorithm

by

Sandra Garcia Esparza, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2009

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Sandra Garcia Esparza

September 11, 2009

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Sandra Garcia Esparza

September 11, 2009

Acknowledgments

I would like to thank my family, specially my parents, Nuri and Jose, for funding my studies and supporting me during all these years, specially this one. I know it is been a hard year, but you have just been great. Also I would like to thank my uncle Carlos, who has always been like my little brother ("'mi tete"') and who can always make me smile. I would also like to give special thanks to my grandfather Joaquin, who decided to invest in my education when I was kid by paying my English classes.

I would also like to thank my friends. To those who have been far but still were close to me. And to those who I've met in Dublin, especially the friends I've met in NDS, for making hard work less hard and without whom this year wouldn't had been such a great experience.

I would also like to thank my supervisor Rene Meier, for guiding me through this project and Jordi Pujol, developer of PlanetSim, for his advice and support.

In general I would like to thank all the people I love. Without them any of this would make sense.

Sandra Garcia Esparza

University of Dublin, Trinity College

September 2009

JXTA-Sim
A simulator for evaluating the JXTA Lookup Algorithm

Sandra Garcia Esparza, M.Sc.
University of Dublin, Trinity College, 2009

Supervisor: Rene Meier

Over the last years the interesting features of P2P networks such as self organisation, scalability and robustness have captivated the interest of researchers. One of the main study focus is on the mechanisms to find resources in these kind of networks. JXTA is a platform that allows developers to build P2P services and applications. JXTA's search algorithm is an hybrid approach between a *Distributed Hash Table (DHT)* and a *Random Walker*. The algorithm makes use of the information in the DHT when the network is stable and only uses the walker when it is unstable.

Previous work has been done to evaluate the JXTA's search algorithm by performing experiments on testbeds. However, most of the experiments have been performed in scenarios with a small number of peers. Although testbeds offer accurate results, performing experiments with a large number of nodes is difficult and expensive. Furthermore, if we want to perform experiments by modifying different parameters, testbeds are not the best approach. Simulators offer a good alternative to testbeds in these situations; they allow to write a model of an algorithm, protocol or system and study their behaviour in large scenarios by performing experiments involving

different parameters.

This project presents JXTA-Sim, a simulator for studying and evaluating the JXTA's lookup algorithm. The aim of this project is to allow researchers using the simulator, to study and evaluate an hybrid search algorithm under different parameters and to test applications that are built on top of it. To build the simulator a P2P framework has been used, which provides some of the necessary components for implementing P2P overlay algorithms.

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Motivation

Over the last years the interesting features of P2P networks such as self organisation, scalability and robustness have captivated the interest of researchers. JXTA is a complex platform consisting of a set of 6 open protocols that allows developers to build peer-to-peer (P2P) services and applications. The JXTA project was originally conceived by Sun Microsystems Inc. and was designed with the participation of experts from academic institutions and industry. Since its creation in 2001, JXTA has been used for commercial and research applications.

The goal of JXTA is to allow developers to create interoperable and scalable P2P services and applications in a distributed environment in an easy way. Although JXTA has a widespread usage in research and industry its performance and scalability still remain unclear. Therefore, we need to find ways to evaluate the JXTA platform.

JXTA is a big platform and therefore its evaluation requires the study of different parts separately. One interesting aspect to study of JXTA is the lookup algorithm and the discovery and routing protocols involved.

Structured overlays, such as Pastry, Chord or CAN, need to store information about the logical graph to be able to place and locate resources. This information is commonly stored in Distributed Hash Tables (DHTs) where keys are assigned to data items and where each key maps to a node in the graph. Using keys to find resources makes the search mechanism easier. However, their maintainance can be expensive, especially in scenarios with high churn

(many nodes joining and leaving the network). Unstructured overlays, such as Gnutella, place resources at random nodes making the search mechanism more "ad-hoc" but at the same time more complicated.

Project JXTA proposes an hybrid approach by combining both approaches [Mohamed et al., 2003]. JXTA's lookup mechanism uses a loosely-consistent DHT when the network is stable and a walker in scenarios where it is unstable and with high churn (ad hoc networks). JXTA's approach is aimed at supporting mobile ad hoc networks where peers join and leave the network frequently. Many recent applications are implemented for these kind of scenarios, which makes the JXTA approach worth studying.

Past evaluations [Halepovic and Deters, 2005], [E. Halepovic, 2004] [Halepovic and Deters, 2003b] and [Halepovic and Deters, 2003a] have studied different aspects of the platform based on benchmarking. However these experiments were performed in testbeds with small number of peers (up to 32 super peers). When evaluating peer to peer algorithms it is very important to test its scalability and 100 nodes is not usually enough. Furthermore, the nodes in the mentioned experiments had the whole JXTA platform installed on them which makes more difficult to evaluate the different parts individually, since they can influence each other.

Although testbeds offer accurate results, testing P2P protocols on large distributed systems is a troublesome, time-consuming and expensive task.

Simulations offer an alternative to testbeds, that makes easier to test algorithms in large scenarios. Simulators are especially suitable when we want to evaluate an algorithm by modifying different parameters or when we want to do modify the algorithm (create a new one or improve an existing one). In this sense, simulations allow us to write a model of an algorithm, protocol or system and study their behaviour through different experiments which can include a large number of nodes.

1.2 Project Aims

The aim of this project is to create a simulator that will study and evaluate the JXTA's lookup algorithm, which uses the JXTA discovery and rendezvous protocols. The JXTA simulator will allow to test the algorithm's scalability and search performance in different scenarios.

This aim will be reached by pursuing three different goals.

The first goal of this dissertation is to do a thorough study of the JXTA protocols involved in the algorithm for publishing and finding resources. At the moment this is being written, there are no books that explain in detail the internals of JXTA. An explanation of the JXTA lookup algorithm can only be found on a few papers and by reverse engineering the complex and extense JXTA code.

Secondly, a model of the algorithm will be implemented in a simulator for its evaluation. This phase includes the study of different P2P frameworks to implement the JXTA lookup algorithm and its implementation in the chosen simulator.

Finally, an evaluation of the algorithm will be done by running different tests. The results of the simulation will show the behaviour of the algorithm in situations with different churns and network sizes.

1.3 Project Contributions

One of the main goals of JXTA is to create scalable P2P applications. However, the performance and scalability of the JXTA protocols still remain unclear. Current evaluations of the JXTA project are based on benchmarking and the experiments have only been performed with a small number of nodes. Although testbeds offer more accurate results they are not very suitable if our goal is to improve an algorithm since a change in the algorithm involves changing it in all the nodes.

In this sense, simulations can help developers to improve their algorithms and compare it with others.

This project aims to study a portion of the JXTA protocols and contribute in the evaluation of this platform and its suitability to create P2P applications.

Although this project will only simulate a part of JXTA, it can easily be extended to simulate and evaluate other parts of the platform. Having a model which can simulate the behaviour of JXTA in a network with a large number of nodes can allow developers to customise and improve the algorithms used to implement the set of JXTA protocols.

Creating a simulator for JXTA's lookup and routing algorithm will not just help JXTA developers to improve their algorithms but also researchers in the field of P2P overlay protocols, who could use it and try to improve the algorithm for other applications that could be benefited

by an hybrid approach between a structured and an unstructured network.

JXTA's hybrid approach is aimed at supporting mobile ad hoc networks where peers join and leave the network frequently. For this reason, researchers working in the field of ad hoc networks could be benefited from JXTA-Sim. Planetsim, the framework which JXTA-Sim is built on, allows to test applications on top of p2p overlay networks. Researchers developing adhoc applications (i.e. p2p applications for MANETs) can use the JXTA simulator to test their applications on top of the JXTA overlay network to evaluate their performance and compare it with other overlay algorithms, such as Chord, which are also implemented in the Planetsim framework.

1.4 Dissertation Outline

This research project is organized as follows:

Chapter 2 offers some background on peer to peer networks, JTXA and overlay searching squemes. It also describes in detail the lookup algorithm used in JXTA which has been implemented by the JXTA simulator.

Chapter 3 presents some related work in the area of simulation of P2P overlay algorithms and presents some research done to evaluate different aspects of JXTA. The last section of this chapter is focused on the simulation framework used for building JXTA-Sim which is PlanetSim.

Chapter 4 covers the architecture and design of JXTA-Sim. It provides a design based on the PlanetSim design and an explanation of the different components that form the JXTA-Sim architecture. This chapter also describes some of the assumptions that were taken in order to implement the simulator.

Chapter 5 refers to the implementation of the simulator. It explains the packages and main classes that have been implemented and also describes how to run a simulation and what parameters can be configured. The last section explains what results can be obtained from JXTA-Sim.

Chapter 6 evaluates the JXTA-Sim simulator. The first section presents an evaluation of the simulator itself, showing how the correctness of the implemented algorithm was tested. The second and last section describes some of the tests performed with the simulator to evaluate the JXTA lookup algorithm and to show the usefulness of the simulator.

Finally, chapter 7 summarises the work done in the project and comments on the future work that can be followed from this project.

Chapter 2

Background

This chapter provides some necessary background before introducing the state of the art. The first sections give a brief introduction to peer to peer networks and search algorithms. The latest sections explain the JXTA platform and focus on the JXTA's lookup algorithm whose implementation is the main goal of this project.

2.1 Peer to peer networks

[Schollmeier, 2001] defines a Peer-to-Peer (P2P) network as a distributed architecture where participants share a part of their resources (processing power, storage capacity, network link capacity, printers) and where these resources are accessible by other peers directly, without passing through intermediary entities.

Therefore, participants of P2P networks are providers and consumers (clients and servers) of resources simultaneously. This architecture model is different than a client/server model where each entity acts as a client or as a server, but never as both at the same time.

Peer-to-peer networks are overlay networks, that is, networks that are built on top of other networks. Peer-to-peer networks build logical networks (or logical graphs) on top of physical networks. The resources on these logical networks are identified by Global Unique IDentifiers (GUIDs). Examples of these resources are peers, groups of peers, files, communication channels or services.

Over the last years the distributed nature of P2P networks has captivated the interest of researchers. P2P offer desirable features like redundant storage, self organisation, scalability and

robustness.

When we have data distributed over a large number of nodes we need search mechanisms to be able find them. The following section explains how we can classify P2P networks according to its structure and the search approaches that can be used for each case.

2.1.1 P2P Overlay Classification

Structured vs Unstructured

P2P overlays can be classified in two groups according to the logical graph structure: structured and unstructured. If the graph follows a specific structure (e.g. hypercub, mesh, butterfly network, de Brujin graphs) then the overlay is structured. On the other hand, if the links between nodes are formed arbitrarily (the graph does not follow a particular structure) then the P2P network is said to be unstructured. Gnutella [gnu, 2009b] is an example of this kind of network.

Structured networks can be sub-classified into highly structured and loosely structured. In highly structured networks, both the P2P topology and the placement of the files are precisely determined whereas in loosely structured networks, the placement of files is based on hints. Freenet [Clarke et al., 2001] and JXTA are an example of this kind of networks.

A core protocol in P2P networks is finding resources. Resource lookup depends on how the data and the network are organised. In structured networks, resources are placed at specified locations making the search process easier. On the other hand, in unstructured overlays, files are placed at random nodes making the search mechanisms more "ad-hoc". These mechanisms usually use flooding or random walk strategies to discover resources.

Resource Indexing

To find resources on an overlay network, data has to be identified using indexing. There are three ways of indexing data:

Centralized Indexing: A central server keeps the indexes of the data stored in the peers. When a node needs to find a resource, it has to issue the query to the central server. Obviously, this approach has a single point of failure and doesn't scale very well as bottlenecks can easily happen on the central server. This type of indexing was used in Napster which used a central

directory lookup.

Distributed Indexing: In this approach the indexes are distributed across different peers in the network. In order to access the indexes a structure needs to be used. The most common mechanism is using a Distributed Hash Table (DHT). Distributed indexing is challenging and different DHT schemes such as Pastry, Tapestry, Chord and CAN have been proposed.

Local Indexing: In this approach each peer only indexes local data and the remote objects that need to be searched for. This approach is usually used in unstructured networks in conjunction with flooding or random walk search.

Hybrid Overlay

DHTs are a structured solution which given a key of a file finds its location. DHTs however, are not always the best solution. They work very well for finding rare files but for situations like file-sharing where most queries are of popular files DHTs would need a lot of extra work, making the search process slower. Another situation where DHT schemes do not perform well is in unstable ad-hoc networks with a high churn rate (nodes continuously joining and leaving the network). If the structure of the network is constantly changing, then updating the DHT will be expensive.

A good solution would be to use DHT when the network structure is stable and use other mechanisms like flooding or walkers otherwise. This approach is used by JXTA in what is called a *Loosely-consistent DHT Rendezvous Walker* [Mohamed et al., 2003]. This is explained in detail in section 2.4.

2.2 Introduction to JXTA

JXTA is a set of protocols designed for ad-hoc, pervasive, peer-to-peer computing. Project JXTA is an open-source project conceived by Sun Microsystems, Inc. in 2001 and designed by the partition of experts from academic institutions and industry. The goal of this project since then has been to standardize a common set of protocols for building P2P applications. JXTA is designed to be independent of programming languages, system platforms, service definitions and network protocols. Furthermore, the protocols have been designed with the goal of being implementable in a wide range of devices including desktop computers, PDAs, sensors, consumer

electronics, network routers, data-center servers and storage systems.

JXTA allows to create a virtual network overlay on top of the existing physical network infrastructure. This virtual network makes possible for peers to exchange messages with other peers independently of their location. JXTA's logical addressing model assigns a unique JXTA ID to every resource in the virtual network. This allows to uniquely identify, not only every peer on the network, but also groups of peers (*peergroups*), communication channels (*pipes*) and *advertisements* (representation of network resources).

The entire JXTA system is modeled using a set of six protocols. JXTA protocols can be implemented using any language, which allows heterogeneous devices to exist and communicate in the P2P network. Current Jxta implementations include Java, C and Perl.

Currently there are six protocols divided into two categories: the textitCore Specification Protocols and the *Standard Service Protocols*. The Core Specification Protocols are the protocols that are required by any implementation that wants to be JXTA compliant. The *Endpoint Routing Protocol* and *Peer Resolver Protocol* belong to this category. The rest belong to the Standard Service Protocols category and are used for devices with larger capabilities. The following list shows the six protocols and gives a brief explanation about their functionality:

Endpoint Routing Protocol (ERP): Is the protocol used to discover a route from one peer to another. For instance, if peer A needs to send a message to peer C and there is no direct route between A and C, the ERP can be used to find the necessary routing information to build a route and send the message.

Peer Resolver Protocol (PRP): The PRP is the protocol used by peers to send queries and receive responses to queries. It is used by the standard service protocols to send and propagate requests.

Rendezvous Protocol (RVP): This protocol is used to propagate messages in the network. The RVP is used by the PRP to propagate messages.

Peer Discovery Protocol (PDP): The PDP is the protocol used by a peer to send a query to one or more peers and to receive a response (or multiple responses) to the query. It uses the PRP to send and propagate the query.

Peer Information Protocol (PIP): This protocol is used by peers to obtain status information about other peers such as the peer's name, how long the peer has been active, how much data has been transferred (traffic load), etc.

Pipe Binding Protocol (PBP): Is the protocol used to create a communication channel (*pipes*) between peers.

2.2.1 JXTA Architecture

The JXTA architecture is divided into three layers as shown in figure 2.2.1.

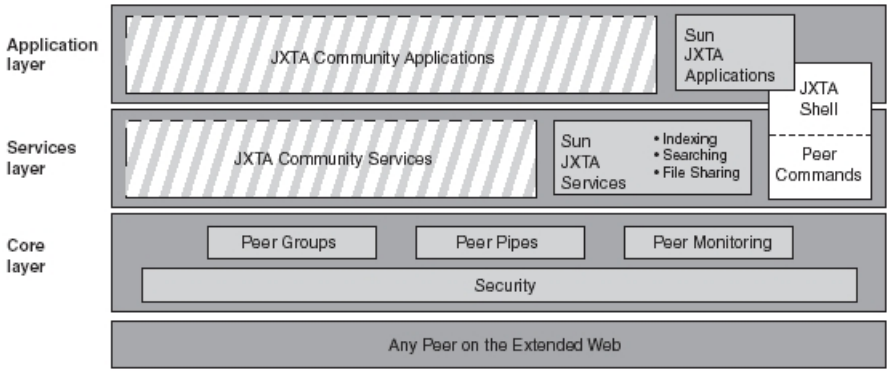


Figure 2.1: JXTA Architecture [Gradecki, 2002]

Core Layer This layer includes the essential primitives to build a P2P network. It is where the code for implementing the JXTA protocols is found. These protocols the functionality to create peer groups, pipes as well as security and peer monitoring primitives.

Services Layer Within this layer we find services created using the protocols of the core layer and that provide functionality which is not essential but is desirable and convenint. These services include searching and indexing, file sharing and membership and authentication services and other JXTA services.

Application Layer This layer provides the components to build applications that use the JXTA services. An example could be an instant messaging application that uses the membership service to join and authenticate users in the network.

2.2.2 JXTA versions and language bindings

Jxta protocols are designed to be independent from the programming language. Current implementations of the JXTA protocols include Java, C, Perl and Ruby among others.

There are two different versions of the Java JXTA ; one for J2SE (Java 2 Standard Edition) and a lighter version for J2ME (Java 2 Micro Edition). The latter is a version for small devices such as phones, PDAs, sensors and other devices. The API for J2ME only implements some parts of the JXTA protocols. This project has been done using the API for J2SE.

The implementation used as reference in this project is JXTA 2.6 for J2SE.

2.3 JXTA Concepts

The aim of this section is to describe the terminology used by JXTA and to explain its main components.

2.3.1 JXTA Virtual Network and Addressing

The JXTA network is a virtual network overlay which lays on top of a physical network. This allows any peer to communicate with any other peer independently of its physical location, overcoming firewalls and NATS by using a logical addressing model. JXTA's logical addressing model assigns a unique JXTA ID to every resource in the virtual network: peers, peergroups, pipes and services. The reference implementation uses 128-bit UUIDs.

2.3.2 Peers

Peers are the basic component in JXTA. A peer is a node with the ability to communicate with other nodes. These nodes can be desktops, laptops, PDAs, mobile phones, sensors, etc. Peers, as the other JXTA components, are identified with a unique 128-bit identifier.

JXTA defines three different kinds of peers:

Edge Peers: They are usually the most common type of peer in a JXTA network. Edge peers can publish and discover advertisements as well as replying to discovery requests by using their local cache. However, edge peers can not forward discovery requests. The responsibles for that are *Rendezvous Peers*.

Rendezvous Peers: They have the same abilities of edge peers, plus they can also forward discovery requests to help other peers find resources in the network. The request will be forwarded within the group of rendezvous until the resource is found or it exceeds a Time To Live. The concept of a rendezvous peer is similar to the concept of a *super node* in P2P networks such as Gnutella 0.6 [gnu, 2009b] or Skype.

Rendezvous peers can be connected to as many other rendezvous as needed. To forward discovery requests, rendezvous peers use a structure called *Rendezvous Peer View* which consists of an ordered list of the rendezvous connections of that peer. During the lifetime of a rendezvous, the entries of this table will vary as other rendezvous join and leave the group.

Each edge peer is connected to a rendezvous peer that they will contact when they need to publish or discover resources. If their rendezvous leaves the network they will try to find another active rendezvous. Edge peers can also become rendezvous peers (be promoted) when they can't find other rendezvous in the network.

Relay Peers: Relay Peers are used when there is no direct physical connection between two peers that need to communicate, for example, a firewall can be in the middle of the two peers. In those situations, peers need to contact a relay peer to find a route to the destination.

Relay peers allow to buffer messages when peers are unreachable or temporarily unavailable. Some messages may need more than one relay too get to their destination.

Edge peers are connected to relay peers during an agreed lease period. After that time edge peers can connect to another relay when they need to use them.

2.3.3 PeerGroups

Peers with a common set of interests organise themselves into groups called *peergroups*. A peergroup provides a context to use applications. For instance, within the same JXTA network we could have peers that belong to a chat peergroup, peers that belongs to a file sharing peergroup and peers that belong to a group that execute computationally intense operations.

Peers can belong to more than one peergroup at the same time. Groups allow to define

different search scopes in the way that a resource published within a certain group can only be discovered by peers that belong to the same group. A similarity can be made with Virtual Private Networks (VPNs), where a computer can talk to another without allowing the rest of the network to participate in the communication.

2.3.4 Advertisements

An *advertisement* is an XML document that describes a JXTA resource. These resources can be peers, peergroups, services or pipes.

When a peer publishes a resource it publishes the advertisement to one if its peergroups. Advertisments are stored in the cache of the peers that publish the advertisement or that have discovered it.

Advertisements are published with a *lifetime* and an *expiration*. While the lifetime specifies the duration (in milliseconds) of the advertisement in the network, the expiration specifies the duration of the advertisement in the cache. Lifetimes allow to flush advertisements that have expired without the need of a centralised management.

Listing 2.1 shows an example of an advertisement describing a JXTA service.

Listing 2.1: Example of a Module Class Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:MCA>
<jxta:MCA xml:space="default" xmlns:jxta="http://jxta.org">
        <MCID>
                urn:jxta:uuid-F9BA645A4F864B9ABDFB9E642D4490DA05
        </MCID>
        <Name>
                Service Example
        </Name>
        <Desc>
                This is the description of an example of a service
        </Desc>
</jxta:MCA>
```

Storing Advertisements

A textitcache management system is used to store information about the JXTA network. Generally, this information are advertisements created by peers or advertisements found during discovery. A directory called *cm* is called when a peer is executed and it represents the local cache of advertisements for that peer.

2.3.5 Pipes

Pipes are virtual communication channels used to send messages between peers and implemented on top of transport protocols such as TCP/IP. Pipes have one input end and one or more output ends. As the other types of JXTA resources, pipes are published and discovered by using advertisements.

2.3.6 Services and Modules

Peers and peergroups can publish and discover services and modules. Each of the JXTA protocols has a service that performs the functionality of the protocol. For intance, the discovery service is used by peers that want to publish or discover an advertisement. At the same time, the discovery service uses the rendezvous service to propagate the request and response messages.

Modules are similar than services in the way that they offer a functionality to other peers. The difference is that a module refers to a piece of code that needs to be downloaded and executed to be able to use it.

2.3.7 The JXTA Protocols

The goal of this section is to explain a bit more in detail the six protocols of JXTA, paying special attention to the protocols simulated in this project: the *Peer Discovery Protocol* and the *Rendezvous Protocol*.

The Discovery Protocol

The Peer Discovery Protocol (PDP) is responsible for discovering and publishing advertisements. The discovery service provides an asynchronous mechanism to discover Advertisements using the resolver service and the Shared Resource Distributed Index (SRDI). As previously seen, each

peer in JXTA has a local cache of advertisements. When a peer finds an advertisement, this is stored locally. Because of this, there are two types of discovery: local and remote. The local discovery uses the cache to find advertisements while the remote discovery uses the resolver service. Discovery takes places within the scope of a peer group. That is, if a peer does a search, it will only see advertisements from peers that are in the same group. For this reason, we need to have an instance of the class PeerGroup to perform a search.

The Resolver Protocol

The Peer Resolver Protocol (PRP) is used to send a query to another peer and receive a response. The PRP is a set of simple messages designed to provide a common messaging system among peers in the JXTA network. There are two type of messages: Query Messages for sending a query and Response Messages to reply to the query. Both are wrappers of XML documents. The Java binding of the JXTA specification builds a resolver service using the PRP specification. This service is associated with a peer group as a core service.

To reduce the amount of processing a service has to do, the messages are delivered to a specific handler on the peer. The handler is a name assigned to a definition that specifies the format of a message, as well as the response that can occur when a message of that type is received [Gradecki, 2002].

Query messages are not guaranteed to reach their destination nor are the results in the Response Message guaranteed to arrive back at the querying peer. The rendezvous may refuse or fail to transmit either message or the answer may not exist. There is also no guarantee of an answer or even a notification that there is no answer.

The discovery service relies on this service to manage the query and response messages necessary to publish and find resources.

Rendezvous Protocol

The Rendezvous Protocol is designed to propagate messages between peers within a group. Rendezvous peers are peers that have agreed to cache advertisement indices, that is, pointers to edge peers that cache the corresponding advertisement. This is a change made in JXTA version 2.0 [Ahkil et al., 2003]. In JXTA 1.0 [Traversat et al., 2002], rendezvous cached the

advertisements, adding more traffic to the network.

Another improvement of version 2.0 to reduce network traffic is that only rendezvous peers are involved in the propagation of advertisement queries. Edge peers are no longer involved in the process of propagating queries. To propagate queries to other rendezvous, each rendezvous maintains a list of IDs of the known rendezvous. This list is called *Rendezvous Peer View (RPV)*. The consistency of the RPV is not enforced; a rendezvous may temporarily or permanently have an inconsistent RPV. To update the RPV of rendezvous, each rendezvous periodically selects a set of rendezvous of their RPV and sends it to their known rendezvous. If a rendezvous cannot find any other rendezvous it can retrieve rendezvous information from a seeding rendezvous. Each group can define its own set of seeding rendezvous.

Endpoint Routing Protocol

The Endpoint Routing Protocol (ERP) is the protocol that allows to communicate two peers when there is no direct route between them. For instance, a firewall can be between of them. In these situations, peers need to contact a relay peer and request a route to the destination. A peer can have multiple peers as relay peers and they can also promote themselves to relay peers if needed.

The ERP defines defines the format of the request and query messages that are processed by relay peers. Relay peers have a routing service that allows them to send a message to the desired destination. Relay peers have local chaches where they store routes that have previously found. When a peer requests a route, relay peers look on their cache to see if they have the route. If the route has expired or is not in the list, they will need to find a new route to the destination. Relay peers can also bridge different physical or logical networks allowing peers behind NATs to communicate with other peers.

Sometimes a message will need more than one relay peer to get to its destination. At any point during the sending of the message the routing information stored in the relay peers may become obsolete requiring relays to discover a new route in order to complete the message delivery.

Peer Information Protocol

The is the protocol by which a peer can request information about another peer. This information can be the time a peer has been active, the traffic sent and received, etc. This is useful if we want to monitor remote peers activities and measure different aspects from the network.

Pipe Binding Protocol

The Pipe Binding Protocol is used by peers to communicate with other peers. Pipes are virtual communication channels between two endpoints. The input endpoint is called *Input Pipe* and corresponds to the receiving end; the output endpoint is the *Output Pipe* and corresponds to the sending end.

There are three kind of pipes:

Unicast These pipes are also called *point to point pipes*. They connect two peers and allow the communication in one way. To send a message the sending side needs to create an output pipe and the receiver side needs the input pipe of the same pipe.

Secure Unicast They have the same features as unicast pipes but they also add security.

Propagate Propagate pipes allow to send a message from one peer to multiple peers. The sender side needs to create an output pipe and each of the receivers will need to create an input pipe.

To be able to use pipes they have to be published by one of the endpoints by using advertisements.

2.4 JXTA Lookup Algorithm

The goal of this section is to describe the JXTA algorithm used for searching advertisements and routing queries which has been implemented and simulated in this project.

The JXTA project does not specify how the search of advertisements is performed but provides a generic resolver protocol framework with a default policy that can be overwritten. Developers can modify the resolver implementation according to their application domain re-

quirements [Ahkil et al., 2003]. The default resolver policy in JXTA 2.0 is based on Rendezvous super-peers and is the one that has been implemented in this project.

This information about the algorithm in JXTA 2.0 is based on the information found in the literature ([Ahkil et al., 2003], [Mohamed et al., 2003], [Theodoloz, 2004], [Antoniu et al., 2007]) and by reverse engineering JXTA's code. It is important to note that at the time this is being written no books explaining in detail this algorithm were found. Doing reverse engineering on JXTA's code is complex due to the large amount of packages and classes and to the little documentation of the code.

2.4.1 Algorithm Overview

Distributed Hash Tables (DHTs) provide the most efficient mechanism to access data (potentially a single hop access). However there is maintenance cost associated that grows exponentially as the churn rate increases. On the other hand, not having DHTs means that we need mechanisms such as network flooding or a walker that are more expensive but do not have a maintenance cost associated.

JXTA tries to combine the advantages of each approach by using a hybrid approach that combines the use of a *loosely-consistent DHT with a limited-range rendezvous walker*. If the rendezvous churn rate is very low the loosely-consistent DHT will be used. On the other hand, if the network suffers many changes, the information may not be found using the DHT and a walker will need to be started instead.

There are two kind of nodes involved in the JXTA lookup algorithm: edge peers and rendezvous peers. Edge peers are responsible for publishing and discovering advertisements. These advertisements are stored in their local cache managed by a *Cache Manager*. Rendezvous peers however, do not store advertisements, but instead they store the indexes of those advertisements. This is an improvement introduced by JXTA 2.x over version 1.x to reduce the amount of traffic in the network.

Rendezvous peers form a rendezvous network where they propagate the queries and responses generated by peers. To keep the connections with other rendezvous, each rendezvous maintains a *Rendezvous Peer View (RPV)*, which is just an ordered list of IDs of the known rendezvous in the group. This list does not have to be consistent. A rendezvous may not know about the

existence of other rendezvous and RPVs can be different on different rendezvous. This is why it is called a *loosely-consistent DHT*.

2.4.2 Advertisement Indexing and Publication

When an edge peer wants to publish an advertisement they push their index to their connected rendezvous. To do so, they use an indexing service provided by JXTA that allows edge peers to index their advertisements. This service is called the *Shared Resource Distributed Index (SRDI)*.

The rendezvous that receives a publish request, stores the index of the advertisement and computes a function to decide which rendezvous should store the advertisement. The function in listing 3.1 shows the pseudocode of this function. `expression` would correspond to the key of the advertisement (this could be the identifier of the advertisement, its name or a combination of both) and `rpv` would be the the rendezvous peer view of the peer calling the function. `sizeOfHashSpace` corresponds to the maximum value returned by the hash function.

Listing 2.2: JXTA getReplicaPeer Function

```
function getReplicaPeer(expression, rpv)
    var Longint digest, sizeOfSpace, sizeOfHashSpace
    var Integer pos
    var JxtaID pid

        digest := jxtaHash(expression)
        sizeOfSpace := length(rpv)
        sizeOfHashSpace := shiftLeft(1, 8 * length(digest))
        pos := (digest*sizeOfSpace)/sizeOfHashSpace

        pid := rpv[pos]

    return pid

{
where:
    jxtaHash(expression)  returns the digest of the given expression
    shiftLeft(n, m) := floor(n * 2m)
}
```

Jxta allows to use replication to increase the locality of the index. The replication is defined

by a *replication distance* which indicates how many peers will store a replica of the index. If the replication distance is one, the index will be replicated into the lower and upper peer in the peer view of the chosen rendezvous. If it is equals to two, it will be replicated into the two lower and two upper peers and so on. Because the peer view is an ordered list of rendezvous peers the

Figure 2.4.2 shows an example of the publication of an advertisement using a replication distance equals to one. When a peer *e1* publishes an advertisement to rendezvous R2, the SRDI indexes it using a predefined number of keys such as the advertisement name or the advertisement ID. The rendezvous R2 will then compute the hash function of the advertisement to map the index to a rendezvous in its RPV. In this case the function returns 5. Then R2 will push the index of the advertisement to R5 and also to its neighbours R4 and R6 (+1 and -1) in the RPV ordered list.

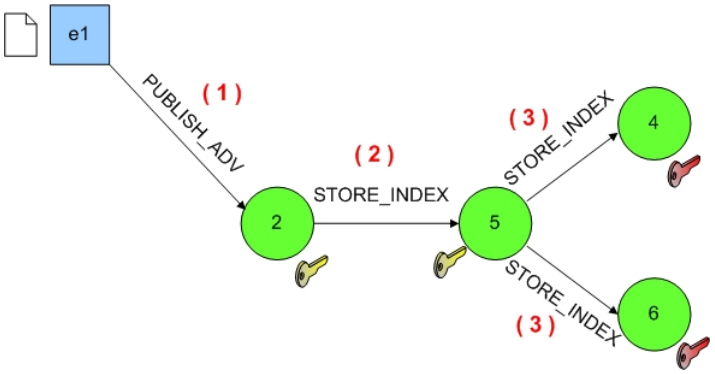


Figure 2.2: Publication and replication of an advertisement

2.4.3 Advertisement Lookup

After the advertisement has been published another edge peer may look for it. The procedure starts in a similar way as in the publication; the edge peer forwards the request to its rendezvous and this will compute the same hash function to choose the rendezvous that should store the index.

As example, let's assume that edge peer *e2* is performing a lookup and its rendezvous is peer 3. From here, we can consider three network scenarios.

1. *a. The network is stable.* If the rendezvous group has not suffered changes since the advertisement was published and R3 has the same RPV as R2, then the hash function

computed by R3 will also return R5. Therefore, R3 forwards the request to R5 which has the advertisement index. Now, R5 forwards it to edge e1 who has the advertisement and which will send to e2.

2. *b. The network suffered some changes.* In this case, R5 has left the network but the other rendezvous still belong to the group. Now, the list in R3 has been shifted, so that now RPV(3)=1,2,3,4,6 and rendezvous 5 is now R6. Because R6 contains a replica it forwards the request to edge peer e1 that will return the advertisement to e2.

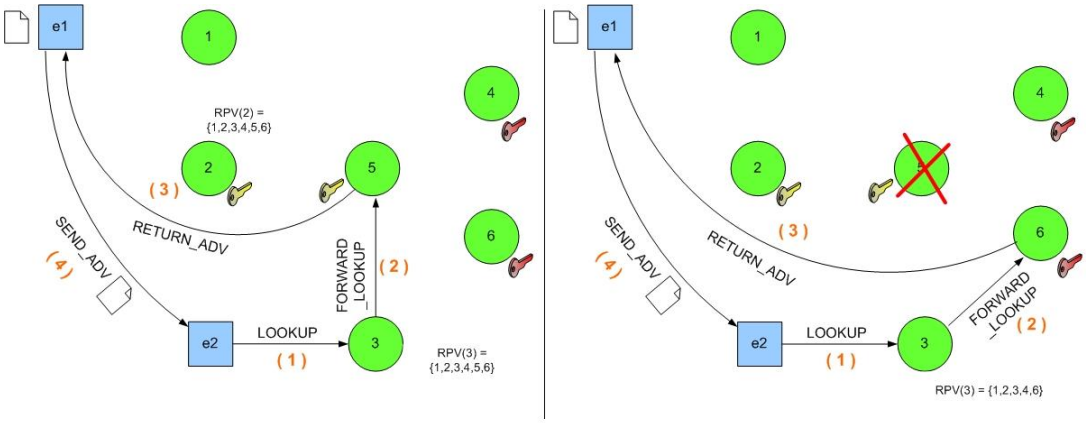


Figure 2.3: Lookup using the DHT (a,b)

The function `getReplicaPeer()` returns the identifier of the rendezvous where the index will be stored and corresponds to the peer whose identifier is closest to the digest of the given expression. The idea behind this is that if the chosen rendezvous has left the next time the function is called, it will return a peer that is close to the previous rendezvous. This way, if the original peer storing the index has left, we can still find a peer that contains a replica.

3. *c. The network is very unstable.* If the network has suffered many changes since the advertisement was published, is possible that the rendezvous chosen by the function `getReplicaPeer()` does not contain the index. In this case, the chosen rendezvous starts a *limited-range rendezvous walker* which proceeds in both up and down directions in the close rendezvous. The walker starts walking in the vicinity of the chosen RPV where the identifier of the rendezvous is closer to the key of the index and this walking continues until the advertisement is found or a maximum number of hops is reached.

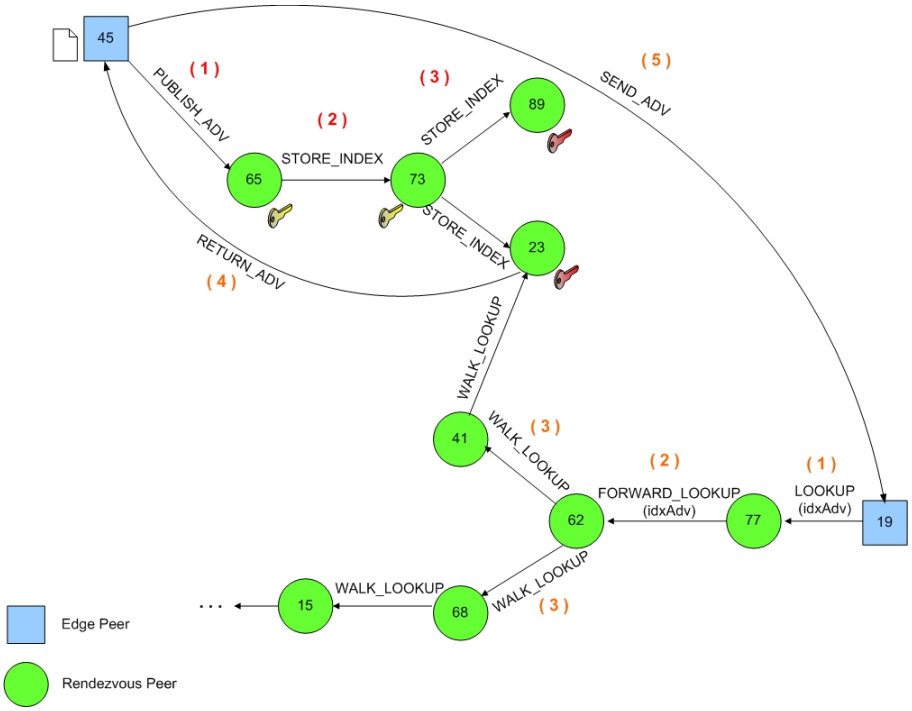


Figure 2.4: Lookup using the walker (c)

It can be observed that the cost of the discovery varies on the scenario with a minimum cost of O(1) if the index is found in the index is found in the first rendezvous and with a maximum of O(M) where M is the maximum number of hops of the walker and M can be less than or equal to the number of rendezvous peers in the network.

2.4.4 Improving data consistency

Query propagation is performed by rendezvous peers by using the information on their RPVs. The problem with routing queries is that routes may change rapidly as the network topology changes. To maintain RPVs consistent, entries are periodically refreshed or flushed.

Rendezvous peers periodically probe other members of the peer view by sending a message with their rendezvous advertisement. When a peer receives a rendezvous advertisement it will refresh the entry for that rendezvous (or add it if it did not exist) and reply with their rendezvous advertisement.

Chapter 3

State of the Art

3.1 JXTA Performance and Evaluation

JXTA is a complex platform composed by 6 complex protocols. These protocols are explained in detail in section 2.3.7. Although it is being used in industry and research projects, JXTA's performance and scalability still remain unclear. However, JXTA is a big platform and studying it requires studying the different parts separately. For instance, some applications do not seem to scale well, but finding what part of JXTA affect its scalability is not simple. We need to find ways to evaluate JXTA's perfomance and scalability.

Past evaluations [Halepovic and Deters, 2005], [E. Halepovic, 2004] [Halepovic and Deters, 2003b] and [Halepovic and Deters, 2003a] have studied different aspects of the platform. These studies are based on benchmarking by performing experiments on testbeds. However, these experiments have been done with a small number of peers (up to 32 super peers) and more focused on the performance of pipes and message throughput.

[Antoniu et al., 2007] have focused on the JXTA discovery and rendezvous protocols, also by performing experiments on a testbed. Although the experiments use a bigger testbed than the previous literature their tests focus on evaluating the consistency of the peerview and on the time it takes to retrieve an advertisement. The first experiment tries to measure the time it takes for the peerview protocol to make all the peerviews consistent. This is, how long it takes until all rendezvous share the same peer view. The experiments show that with more than 45 rendezvous peers the rendezvous protocol fails to make the peer views consistent. However, the

goal of JXTA's lookup algorithm though is to find advertisments even when the peer view is not consistent, which most probably won't be, especially in networks with high mobility and churn.

Also, performing tests using nodes that are running the JXTA platform does not give accurate results about the lookup algorithm because other factors (e.g. traffic caused by other JXTA services) can influence the results.

For this reason, we consider that other experiments need to be done regarding the lookup algorithm.

Although testbeds offer accurate results, performing experiments with a large number of peers is difficult and expensive. Simulations offer an alternative to testbeds. Even they are not as accurate as testbeds, they are usually a better approach when evaluating and comparing algorithms that depend on many parameters. Another advantage is that if we are creating a new protocol, or improving an existing one, making and testing changes on a simulator is very simple. For instance, if we want to change differrent parameters as during our experiments running on testbeds, we must change the settings on each machine, while using a simulator, we usually have one configuration file which makes this process much easier.

Another advantage of using overlay simulators is that usually they allow us to test applications on top of the simulated overlay network.

For this reasons, a simulator has been chosen as the approach to study the JXTA lookup algorithm. The simulator will provide the user with a solution that allows to test and improve the algorithm under different parameters.

3.2 Evaluation of peer-to-peer protocols

Peer-to-peer (P2P) systems have attracted the attention of researchers studying distributed computing. Some prominent research projects include the Chord project, the PAST storage utility, the P-Grid, a self-organized and emerging overlay network and the CoopNet content distribution system. Researchers need ways to ease the design of new protocols and to evaluate their behaviour.

Due to the distributed nature of these kind of systems their study and evaluation is a major challenge. There are different approaches we can use to evaluate P2P systems and protocols. These are shown below:

Crawlers A crawler is a program running on a node that collects data passing through that node. getting a local view of the P2P network. By deploying various crawlers, a bigger view of the network can be monitored and studied. The drawback of this approach is that it can not show a global view of the network because the behaviour of the nodes that do not have a crawler is unknown.

Emulators An emulator duplicates the behaviour of one system using another system, so that the second system behaves like the first system. Within the context of distributed computing, an emulation allows to configure a distributed system in order to reproduce the behaviour of another distributed system. Emulators allow to have a global view of the system but their execution is slow because messages need to be passed between different emulator processes through a network.

Simulators . A simulator allows to define a model of a P2P system or a P2P algorithm and then study its behaviour by running different experiments. By defining a simplified version of a real environment they allow to run faster experiments than by using emulators. Therefore, simulators are well suited to evaluate algorithms that require intensive computations. The downside of this approach is that results are not very accurate in comparison with real world results. Simulations can be run on a single sequential machine or on several parallel machines.

Testbeds A testbed is a platform for running experiments of large-scale environments. The results obtained using this approach are usually the most realistic results. A known example of this approach is PlanetLab, a platform for deploying and testing distributed environments.

When designing a new protocol we believe it is important to write a model of the algorithm and evaluate it before using real testbeds. Overlay simulators allow us to test new P2P protocols and make improviments without having to deploy large and expensive systems. Another advantage is that they allow to easily compare different results obtained by the modification of a parameter (e.g. incrementing the number of nodes in the network).
It is very common to compare a new algorithm to existing ones to evaluate its performance. Simulators allow to easily compare the results obtained by the simulation of different protocols.

3.3 Peer-to-peer Simulators

3.3.1 Network Simulators vs Overlay Simulators

Within simulators we can find network simulators and overlay simulators. Network simulators provide a framework for accurate simulation of network protocols such as TCP, UDP, IP, etc. These simulators model the network at the packet level, considering parameters such as delay, bandwidth and other lower-level concerns. Some well known network simulators are Opnet, NS-2 and OMNET++. These simulators perform very well when evaluating network protocols but they do not scale well for overlay networks with a large number of nodes. For instance Omnet++ can't simulate more than 1000 nodes and Narses can simulate up to 600. This is due to the overhead added by the network details.

On the other hand, overlay simulators are less focused on the lower level and more focused on evaluating the overlay algorithms.

3.3.2 Desirable characteristics of P2P simulators

P2P systems are becoming popular due to their interesting properties such as decentralisation, scalability, self-organisation and robustness. Those properties impose some important requirements on the simulator. The following list shows some of the desirable characteristics a peer-to-peer simulator should have [Naicken et al., 2007], [Baumgart et al., 2007]:

Scalability P2P protocols need to be scalable to thousands of nodes. A p2p simulator should be able to run simulations with a large number of peers while making an efficient use of computing resources.

Flexibility The p2p simulator should be able to run simulations of both structured and unstructured overlay networks. The user of the simulator should also be able to specify the parameters which relevant for a specific simulation such as the number of nodes, the mobility of the nodes or the churn rate.

Usability and Documentation Usability is related to how easy to learn and use the simulator is. A p2p simulator should have a clear and understandable API that allows to implement protocols in an easy way. A good documentation explaining how to use the simulator

is also very important. Some simulators have very poor documentation and the code is difficult to understand which makes these simulators very hard to use.

Underlying Network Simulation Some simulators do not model the underlying network or they offer a limited simulation of the network layer. This makes simulations of peer to peer protocols not as realistic as with simulators with a better modelling of the network layer. In some cases researches prefer to focus on the algorithm verification without worrying much about some parameters of the network layer such as latency costs. In other cases, a proper simulation of the network layer is necessary. In those cases, an exchangeable network model would be desirable.

Statistics The p2p simulator should be able to collect significant results which are easy to understand and manipulate.

Repeatability Mechanisms should exist to allow the repeatability of simulations. Repeatability is important to reproduce experiments, compare different proposals and evaluate the influence of a parameter by changing it in different simulations. Some papers present results that are not reproducible and therefore, comparisons between different proposals are difficult to do.

It is very difficult to built a simulator that satisfies all the requirements. To fill some requirements some simulators need to sacrifice other requirements. For instance, if a simulator wants to offer high scalability probably the network layer will need to be represented by a simple model, without considering the low-level details such as the overhead associated to the communication stack. Sometimes, an accurate level of detail is not necessary to evaluate some protocols.

The difficulties found in satisfying all the requirements have led to different research groups to develop their own simulator to evaluate their protocols.

3.3.3 Study of existing simulators

Although there is a wide range of simulators, most of them are "'home-made"' solutions built to simulate a specific protocol or systems. Studies done in [Naicken et al., 2007] examines 287

Simulator	**P2P Protocols**	**Max Nodes**	**Language**
NeuroGrid	Gnutella	$<$ 300,000	Java
PeerSim	Internal P2P Models	$>$ 1,000,000	Java
P2PSim	Chord, Accordion, Koorde, Kelips, Tapestry, Kademlia	3,000	C++
PlanetSim	Chord, Symphony	100,000	Java
OverSim	Chord, Kademlia, Koorde, Broose, GIA	100,000	NED

Table 3.1: P2P Simulators Characteristics

papers and shows that almost 50% did not state what simulator they used and of the ones they did, 62% used a simulator built for a specific algorithm.

Some of these simulators are built for simulating file sharing systems [Schlosser et al., 2002], [Belmonte et al., 2007] while others such as FreePastry [fre, 2009], Neurogrid[Erich(extern), 2003] and P2PRealm[Kotilainen et al., 2006] are more focused on simulating p2p overlay algorithms.

There are many research papers such as [Naicken et al., 2007], [Wei et al., 2007] and [Brown and Kolberg, that have done a survey about the generic P2P simulators. The surveys compare the different simulators according to some of the desired features seen in section 3.3.2 and show that there isn't a simulator that satisfies all of the characteristics.

The table 3.1 shows a comparison of different simulators according to their scalability and the protocols included in the simulator. This information is based on [Brown and Kolberg, 2006] and [Naicken et al., 2007]. Most of the information, such as the maximum number of nodes, has been obtained from the papers published by the corresponding simulator developers.

Although there are no simulators that satisfy the characteristics explained in section 3.3.2, we have to choose the simulator which suits our needs better. Documentation was an important factor in choosing a framework for JXTA-Sim. Some of the previous simulators, such as P2PSim, have very little documentation which makes them difficult to extend.

OverSim [Baumgart et al., 2007] is an overlay network simulation framework based on OMNet++ [omn, 2009] that allows to simulate both structured and unstructured peer-to-peer protocols. Although OverSim offers good documentation and a flexible network layer based on OMNet++, it uses NED as the language for writing modules, which is not as known to developers as Java or C++.

PlanetSim [Pujol-Ahulló et al., 2009] is a discrete event-based simulator that allows to imple-

ment and validate overlay algorithms and to create and test new services. It is an object-oriented, extensible and customizable framework implemented in Java. In order to provide extensibility, it presents a layered architecture where each element is an easily extendible component. PlanetSim is well documented and the developer team offer very good support which makes it an attractive tool to developers.

For these reasons, PlanetSim has been chosen as the framework for implementing JXTA-Sim. The following section explains some of the concepts of PlanetSim and its architecture, which are needed to understand the design and implementation of JXTA-Sim.

3.4 PlanetSim Overview

PlanetSim [Pujol-Ahulló et al., 2009] is a discrete event-based simulator written in Java that allows to implement and validate overlay algorithms and services. JXTA-Sim has been built on top on of the PlanetSim framework and therefore it has been designed according to PlanetSim's design. The goal of this section is to explain the components and architecture of PlanetSim which have been necessary for building JXTA-Sim.

3.4.1 PlanetSim Architecture

PlanetSim architecture is composed of three layers: the application layer, the overlay layer and the network layer. Each layer includes different components that can be extended to adapt to the algorithms and services that need to be simulated. Figure 3.1 shows the different layers and its components.

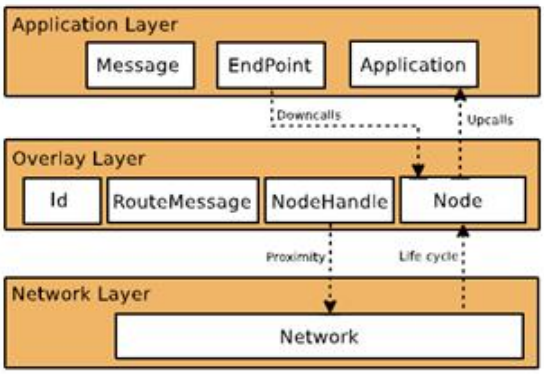


Figure 3.1: PlanetSim Architecture

Each layer consists of a set of objects that can be overridden by the developer to implement the desired algorithm.

Application Layer It is the layer that allows to test applications over different overlay schemes. It consists of three main classes: EndPoint, Application and Message. The *EndPoint* class is responsible for the messages going from the Application to the Node while the *Application* class is responsible for the calls coming from the Node to the Application. The type of these messages can be defined by the developer by extending the class *Message*.

The following figure shows the interfaces of the main classes in the Application Layer.

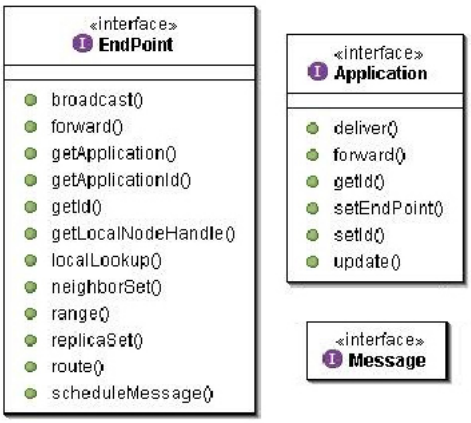


Figure 3.2: Main interfaces in the Application Layer

Overlay Layer The overlay layer represents the virtual layer on top of the physical layer. This is the layer where the behaviour of the overlay algorithm is defined. There are four components in this layer:

- **Node:** This is the main class of the Overlay Layer. The developer uses this class to inject the behaviour of a node by implementing the algorithm of the desired overlay scheme. A *Node* can receive and send messages and perform actions when those events happen. All nodes have a *NodeHandle* used to manipulate the node. They also have an incoming queue and an outgoing queue to store the incoming and outgoing messages.

- **Id:** This class is used to identify nodes in the network. It must be defined according to the algorithm being implemented. For instance, in Pastry and JXTA the IDs are

128 bits identifiers. It can also include information about the node's network address (such as IP/port).

- **NodeHandle:** The *NodeHandle* represents a handler to manipulate a node. It is composed of the identifier of the node and a boolean variable that indicates if the node is alive or not.

- **RouteMessage:** Defines the message sent between the EndPoint and the Node which contains information to route a message, such as source, target and next hop.

Figure 3.3 shows the methods of the main classes in the Overlay Layer.

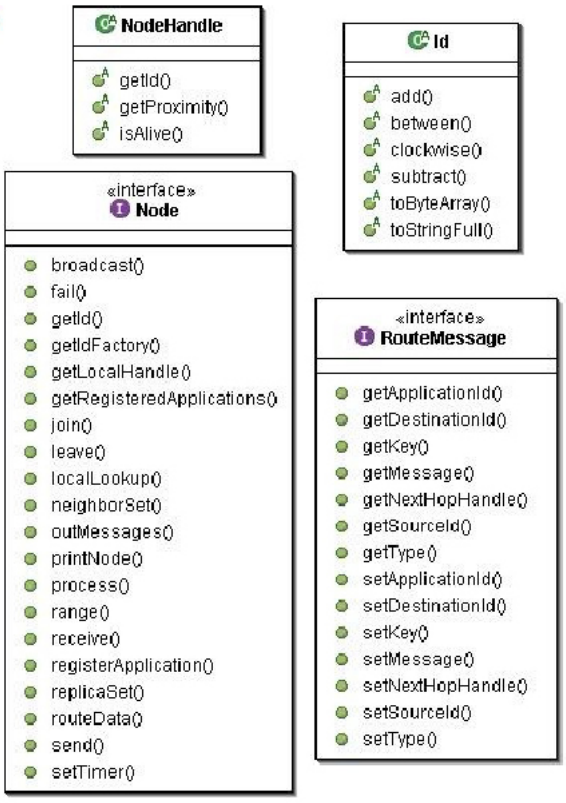


Figure 3.3: Main interfaces in the Overlay Layer

Network Layer The network layer represents the physical network where nodes communicate with each other by sending *Route Messages*. At each step, the simulator checks if there are any messages in the outgoing queues of each node and if there are, they are sent to their destination.

The network layer in PlanetSim is quite simple and it does not introduce details such as

latencies or node mobility. However, it can be extended to simulate more complex networks. The latest paper published by the authors of PlanetSim [Pujol-Ahulló et al., 2009] introduces an extension of the simulator which takes into account latencies. Moreover, the authors mention the possibility of integrating PlanetSim with an existing simulator such as NS2 or OMNET++ to simulate more realistic networks.

We can see the methods of the network class in figure 3.4.1.

Figure 3.4: Main interfaces in the Network Layer

3.4.2 The Network Simulator

One of the most important components in PlanetSim is the *Network Simulator*. The network simulator includes an instance of the Network as well as an event Scheduler. The *Scheduler* includes a set of events which can be loaded from a file and that will be simulated by the simulator at every step.

For every step, the simulator checks the incoming and outgoing queues of each node in the network and handles them accordingly. The messages in the outgoing queue are sent to the destination node by the network, while the messages in the incoming queue are passed to the node by calling the `process()` method. Therefore, when implementing the behaviour of a node, we must place the code to be executed in every step inside the `process()` method.

Parsing event files

There are three kind of events we can simulate in PlanetSim: *join*, *leave* and *fail*. These events can be parsed from a file and stored into a structure that the simulator will read at the corresponding step.

The format of each type of event in the file is the following:

at < num-step> JOIN / LEAVE / FAIL < params> < num-events>

The following code listing shows an example of a simple event file.

```
at 10 JOIN 73289793749 13109375372 1
at 27 JOIN 32424232324 73289793749 1
at 42 LEAVE 73289793749 1
at 55 FAIL 32424232324 1
```

3.4.3 The `Node` object

The `Node` is one of the most important objects of the framework because is the component that contains the behaviour of the algorithm. When implementing a new algorithm on PlanetSim we must extend the class `Node`. As we can see in figure 3.3 this class contains some of the basic methods of a node. However, we can overwrite this methods if we need to. Some of the common methods for a node are the `join, leave` and `fail` methods.

Usually nodes in a network need to execute periodic actions such as probing other nodes. PlanetSim allows nodes to define *tasks* to be executed periodically. For every new task, we can define when it will start executing, and periodicity at which they will be executed.

3.4.4 Results

PlanetSim is a bit limited when it comes to gathering results. However, it provides the package `planetsim.commonapi.results` with some components to generate a graph of the network. This allows to see the nodes present in the network and their connections. The current formats available are GML [gml, 2009] and PAJEK [paj, 2009].

Lately we can also obtain a simple aspect from the developer group which generates a graph in Gnuplot[gnu, 2009a] format that illustrates the different types of messages sent during the simulation.

3.4.5 Configuration of a new algorithm in PlanetSim

PlanetSim uses configuration files to configure a simulator for a new p2p algorithm. To do so it uses two configuration files:

Master Configuration File This file contains a list of all the simulation tests available for PlanetSim and the path of the corresponding configuration files for each simulation test. The following listing shows some entries of this file.

Listing 3.1: Entries in the PlanetSim master configuration file

```
##########################################################
        # This file contains the configuration parameters to test
        # the JXTA algorithm for publishing and finding resources
        #
        JXTA_LOOKUP_TEST = conf/jxta.properties

        ##############################################################
        # This file contains the configuration parameters to test
        # the dht of the Chord algorithm
        #
        CHORD_DHT_TEST = conf/chord_dht.properties

        ##############################################################
        # This file contains the configuration parameters to test
        # the Scribe application on top of a Chord overlay
        #
        SCRIBE_TEST = conf/chord_scribe.properties
```

Specific Configuration File This file is used to configure the parameters of the simulated algorithm. Its content can be classified in different parts. The most important are:

Factories Properties These properties allow to specify the factory for each of the components in PlanetSim. Factories are used to build instances of the different components during the simulation.

Simulation Properties These properties are related to the simulation and contain properties such as the size of the incoming and outgoing queues or the event file to be loaded.

Overlay Properties These properties correspond to the classes used to represent the different components. Here we specify the class used to represent an Id, a Node and the class for the algorithm Properties. This gives a lot of flexibility and modularity, since we can specify what components to be used for each simulator. For instance, for simulating Chord the default Node class will be the `Chord Node`, while for simulating Jxta, the class used to represent the node will be the `Jxta Node`.

Overlay Algorithm Properties Here we can specify the properties that will be read by the corresponding Properties class (e.g. Jxta Properties) and that are particular to the algorithm. For instance, in Chord a property could be the number of steps to fix the finger table, while in Jxta a property could be the maximum number of hops of the walker.

Results Properties These properties correspond to the classes used to obtain the results after a simulation.

Chapter 4

JXTA-Sim: Design and Architecture

4.1 Jxta-Sim Overview

JXTA-Sim is a simulator that tries to simulate the behaviour of the JXTA lookup algorithm described in [Mohamed et al., 2003]. Some of the details about the algorithm which have not been found in the literature have been obtained from the implementation of JXTA for J2ME, version 2.6.

JXTA-Sim is built on top of PlanetSim, a P2P simulator framework, and therefore its design and architecture depend on PlanetSim's design.

JXTA-Sim tries to satisfy three requisites:

- **Scalability**. One of the most important characteristics in a P2P algorithm is its scalability. JXTA-Sim should allow to study the JXTA lookup algorithm performance in scenarios with a large number of nodes. PlanetSim can simulate up to 100.000 nodes and therefore we should expect a similar scalability with JXTA-Sim.

- **Extensibility**. The current version of JXTA-Sim only simulates the lookup algorithm. However, other Jxta features could be added in the future. For this reason, the design of JXTA-Sim should be simple and easy to extend.

- **Usability**. The simulator should be easy to use and should allow the researcher to configure all the parameters necessary for their tests. Moreover, the simulator should provide results which are easy to understand and significant for the evaluation of the simulated

algorithm.

4.1.1 JXTA-Sim use cases

Figure 4.1 shows the system's use cases. The first step for the user is to configure the simulation. By using a properties file the user can define different parameters used for the simulated JXTA algorithm such as the cache size of edge peers used to store advertisement, or the maximum number of hops the walker can perform.

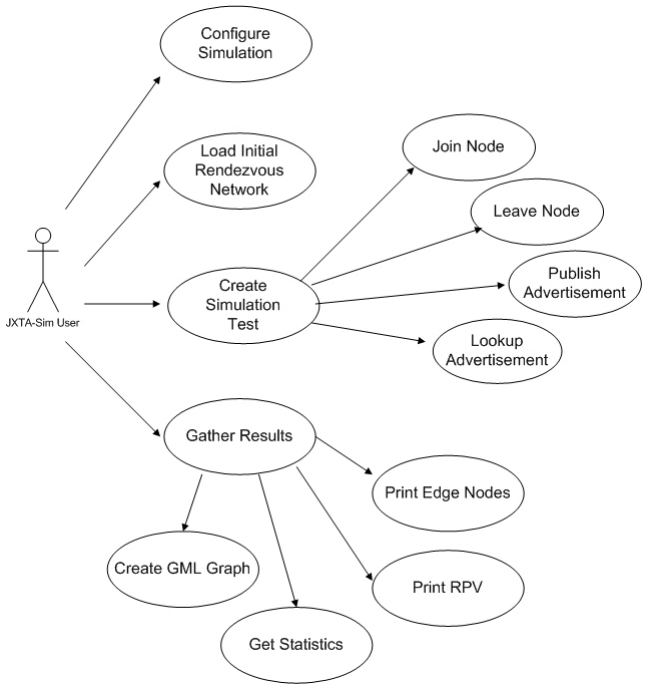


Figure 4.1: JXTA-Sim Use

There are four types of events that can take place during a simulation:

1. **Join** A peer joins the JXTA network. When a peer joins, we must specify another peer to use as bootstrap. In the case of an edge peer joining, the bootstrap is the rendezvous connected to the edge, while in the case of a rendezvous peer, it will be another rendezvous peer in the network. Rendezvous peer views need to be initialized when a rendezvous joins the network. The rendezvous peer joining will request the peer view to the rendezvous used as bootstrap.

2. **Leave** A peer is disconnected from the network. In JXTA-Sim, before a rendezvous peers leaves the network it will need to migrate the edge peers connected to it to another active

rendezvous so the advertisements published by edge peers can be found.

3. **Publish** An edge peer creates and publishes and advertisement. The advertisement is stored in its cache and the index (or key) of the advertisement is published to its local rendezvous peer to propagate it in the network.

4. **Lookup** An edge peer searches for an advertisement. The edge peer sends the key of the advertisement to be found to its local rendezvous that will propagate the query within the rendezvous network.

Events can be simulated by loading a file with the events or by writing a class which creates them. If we use an event file, we can specify what type of events will be executed at a certain step of the simulation. This allows the user to have repeatable simulations.

The user can also write a class that invokes the different types of events. This can be useful if we want to add randomness to simulations. For instance, we can run a simulation where peers are chosen randomly to perform an action. As a result, each simulation will show different results every time. The number of each type of events can be specified by the user. For instance, the user can specify, for a certain simulation period, how many advertisements will be published and how many lookups will be performed. The user can also decide, for that period of time, how many peers will join and how many peers will leave.

Finally, when the simulation is finished, the user has the possibility of gathering different results. One of the results we can obtain is a graph of the network written in GML format that can be later visualized with a graph editor such as yEd [yed, 2009]. By using aspects we can also obtain graphs that show the messages sent by nodes at every step or the number of hops of a search. Finally, we can also obtain how many searches have been successful (the advertisement has been found) and see information about peers such as the rendezvous peers view and the edge nodes connected to them.

4.1.2 Assumptions

To simplify the design of the simulator, some assumptions have been taken into account:

1. *There is only one peer group and all nodes belong to this group.* In JXTA, the discovery and routing of queries is limited to the scope of a group. JXTA allows to have multiple

groups and peers can belong to one or more groups. For simplicity, the current design of JXTA-Sim assumes there will be just one group in the network. This means that all the advertisements published by any peer should be found by any peer in the network, since all the peers belong to the same group.

When doing a simulation we will assume that all the nodes belong to the same group and all the messages are propagated within that group.

The inclusion of peer groups into Jxta-Sim is left for future work.

2. *Rendezvous peers never fail.* Rendezvous peers that leave the network will always do it "on their own will", but never because of a failure. The reason behind this is to simplify the design and the performance of the simulator. This is explained in detail in section 4.3.4.

3. *Edge peers never leave the network.* Edge peers are always connected to a rendezvous peer. If a rendezvous peer leaves all the edge peers of that rendezvous are migrated to another active rendezvous.

   Therefore, if an advertisement is published it will never disappear from the group. However, if the rendezvous leaves it could not be found due to the "migration" process (or, in Jxta, that would be, while the edge peer looks for another rendezvous). In the simulation, this is, looking for an active rendezvous.

4. *Relay Peers are excluded from the simulation.* The description of the algorithm doesn't include relay peers. Therefore, to evaluate the Jxta search mechanism we don't need them. To simplify the simulator design we assume that there aren't firewalls or NATs between peers and therefore relay peers aren't included in the simulations.

4.2 Jxta-Sim Architecture

In order to build JXTA-Sim we have to extend and overwrite some of the components of PlanetSim. The following figure shows the architecture of JXTA-Sim as an extension of PlanetSim's architecture.

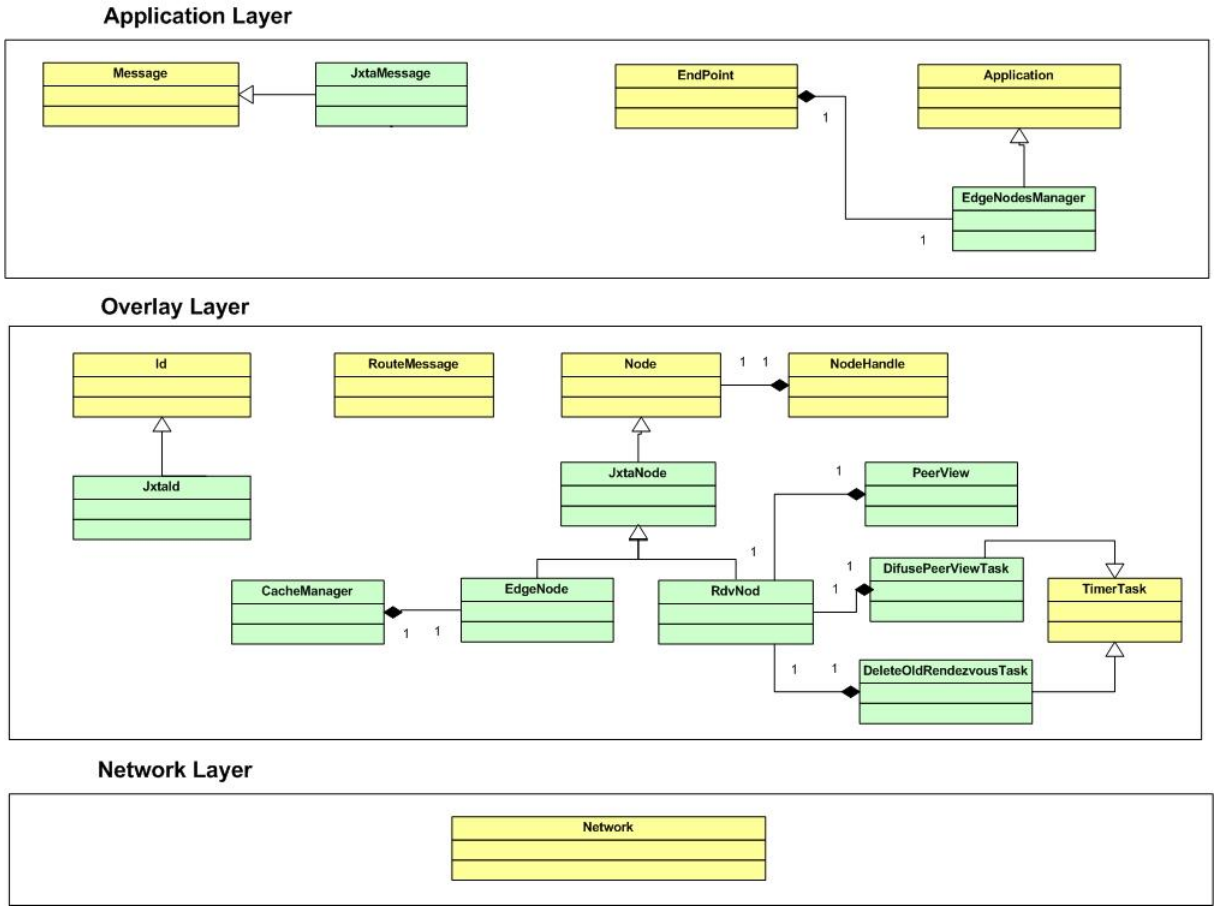


Figure 4.2: JXTA-Sim Architecture

One of the main components is the `Node`. There are two types of nodes that are involved in the JXTA lookup algorithm: *Edge Nodes* and *Rendezvous Nodes*. In PlanetSim we can only specify one type of node for the network. To be able to have two types of nodes we need a `JxtaNode` that extends the PlanetSim `Node` and that is inherited by the objects `EdgeNode` and `RdvNode`.

Edge nodes have an structure to store the published and discovered advertisements. This is the *Cache Manager*. On the other hand, rendezvous nodes need a structure to maintain the connections with other rendezvous. This is the *Peer View*. In order to update the entries of the peer view, rendezvous will periodically execute two tasks: *DifusePeerView* and *DeleteOldRendezvous*. The first task is responsible for periodically sending some of the entries of the peer view to a random set of rendezvous, while the latest is responsible for removing those entries which have expired.

One of the components we have in the application layer is the *Edge Nodes Manager*. This component is used to easily manage a group of edge nodes connected to a rendezvous. Each rendezvous node has an *Edge Nodes Manager* that contains a certain number of `EdgeNode`s.

Nodes in Jxta-Sim communicate using four kind of application messages: *Peer View Message, Discovery Message, Edges List Message* and *Advertisement Message*. These messages are encapsulated in a `Route Message` before being sent on the network. JXTA messages are explained in more detail in section 5.3.

Finally, the lower level component is the `Network`. The network is composed of a set of nodes that execute events at certain steps during the simulation. PlanetSim offers a simple model of the network without considering aspects such as latencies or node mobility. JXTA-Sim uses this network model which is enough for algorithm verification. However, as seen in section 3.4.1, this model can be extended to simulate more realistic networks.

4.3 Simulation of Events

4.3.1 Joining of a Rendezvous Peer

When a peer joins the network it uses another peer as bootstrap to obtain some information about the network. When a rendezvous peer joins the network it needs to initialize its rendezvous peer view (RPV). To do so, it sends a message to the bootstrap node requesting the peer view. The bootstrap node then sends its peer view inside a *Peer View Message*. The user of the simulation can decide the size of the peer view to be sent by changing the parameter MAX_PEERVIEW_SIZE in the configuration file. If the peer has a large peer view we may only want to send some of the entries to minimize the traffic in the network.

To be able to manage their local edge peers, rendezvous peers need to register an *Edge Peers Manager* when they join the network. At this point, though, no edge peers are connected to the rendezvous peer.

4.3.2 Joining of Edge Peers

During the simulation we can add edge peers to a rendezvous. The current version allows to add a random number of edge nodes to a rendezvous peer. This set of edge peers will be managed

by the rendezvous peer through the *Edge Peers Manager*.

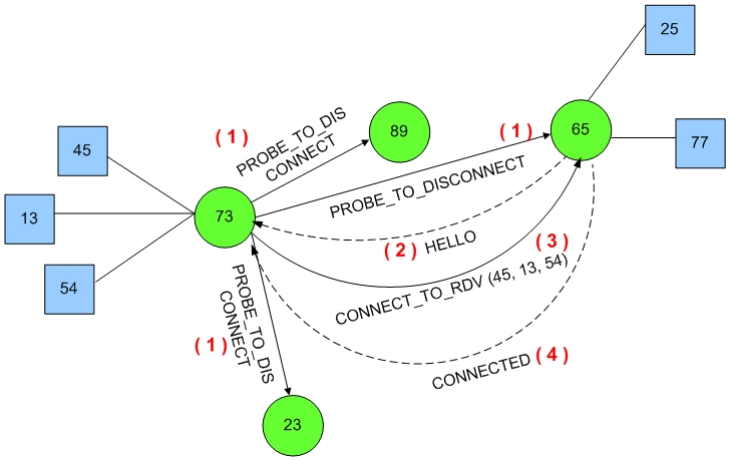
4.3.3 Publishing and Discovering Advertisements

During the simulation, edge peers can publish and discover advertisements by sending a *Discovery Message* to their local rendezvous peer. Rendezvous peers implement the algorithm described in 2.4 to reply to other peers requests. Rendezvous peers will have a message handler to handle different types of messages. By using their peer view, they can forward publication messages to other rendezvous peers.

4.3.4 Disconnection of Rendezvous Peers

When a rendezvous leaves the network or fails, the information on the peers connected to it will be inconsistent. To make this information consistent again, peers periodically probe each other to check their connections are still active. In the case of edge peers, if their local rendezvous leaves the network, they will need to find another active rendezvous before they can publish and discover advertisements. Also, during this time, their published advertisements can not be sent to the originator of the query.

To simplify this in JXTA-Sim, edge peers do not constantly probe their rendezvous. Instead, when a rendezvous leaves the network, before doing so, it needs to find an active rendezvous to connect the edge peers to. This is shown in figure 4.3.4.



The first step is to probe some of the rendezvous in its peer view (1), which are randomly chosen. After receiving a response from an active rendezvous (2) (and ignoring the other responses) it sends a list of its edge peers to the rendezvous (3) in an *Edges List Message*. Finally,

it waits for an acknowledgement confirming the transfer of edge peers (4) before disconnecting from the network. If the acknowledgement is not received it will try to connect to another rendezvous.

Because the messages are always delivered in the current version of JXTA-Sim, if the acknowledgement is not received is because the rendezvous disconnected before. Therefore we can guarantee that there won't be two rendezvous with the same edges connected to them.

4.3.5 Updating the Peer Views

Each entry in the rendezvous peer view has an expiration that is initialized to a default value when the new entry is stored. This value can be configured by the simulator and its decrease by one at every step.

To keep the rendezvous peer views consistent two tasks are defined and executed periodically: *Difuse Peer View Task* and *Delete Old Rendezvous Task*. The simulator needs to allow the user to configure the periodicity of these tasks.

Difuse Peer View Task

When this task is executed the rendezvous will choose a random set of entries from the RPV and send them to some random rendezvous. When a peer receives a RPV it refresh the expiration of that entry to the default expiration value. After refreshing those entries, the rendezvous peer sends a message to the sender of the peer view to confirm that it is alive. On reception of this message, the rendezvous updates the entry of that peer.

Delete Old Rendezvous Task

To save storage capacity in the nodes, expired entries have to be removed when they haven't been updated after a period of time. When this task is executed, it will check the expiration times of each entry and if it is equals or less than zero it will remove the entry.

Chapter 5

JXTA-Sim: Implementation

This chapter explains the implementation of JXTA-Sim by using the PlanetSim framework which provides the components needed for creating a simulator for a p2p algorithm. The version used for this project is PlanetSim 3.0. This chapter explains how the JXTA lookup algorithm has been implemented according to the description in section 2.4 by following the design described in chapter 4.

5.1 JXTA-Sim Packages and Classes

Planetsim main package is `planet` and its main packages are `commonapi` which includes the main interfaces for building an overlay algorithm and `generic.commonapi` which includes the implementation of these interfaces. Those interfaces correspond to each of the components of the PlanetSim architecture shown in figure 3.1.

JXTA-Sim is integrated with Planetsim by adding a new `jxta` package to the project. This package includes the packages: `message, cm, rpv` and `util` as well as the classes that implement the JXTA nodes and identifiers.

The following diagram shows the main packages of PlanetSim and the packages created for JXTA-Sim.

Sections 5.2 explain the main classes in the `planet.jxta` package while section 5.3 describes the different types of messages sent during a JXTA-Sim simulation and the classes used to represent these messages.

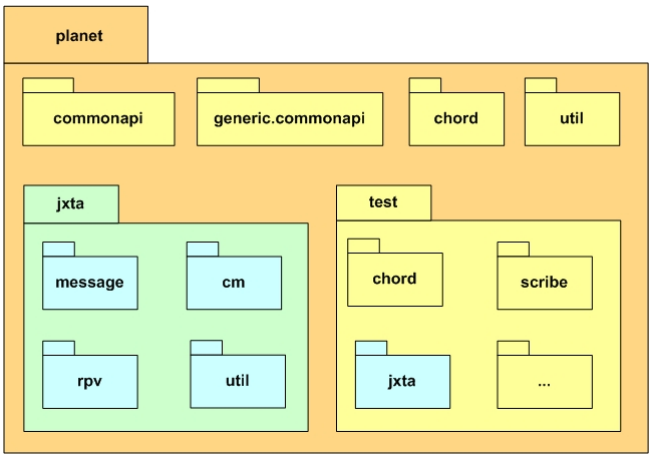


Figure 5.1: JXTA-Sim packages integrated inside `planet` package

5.2 The package `planet.jxta`

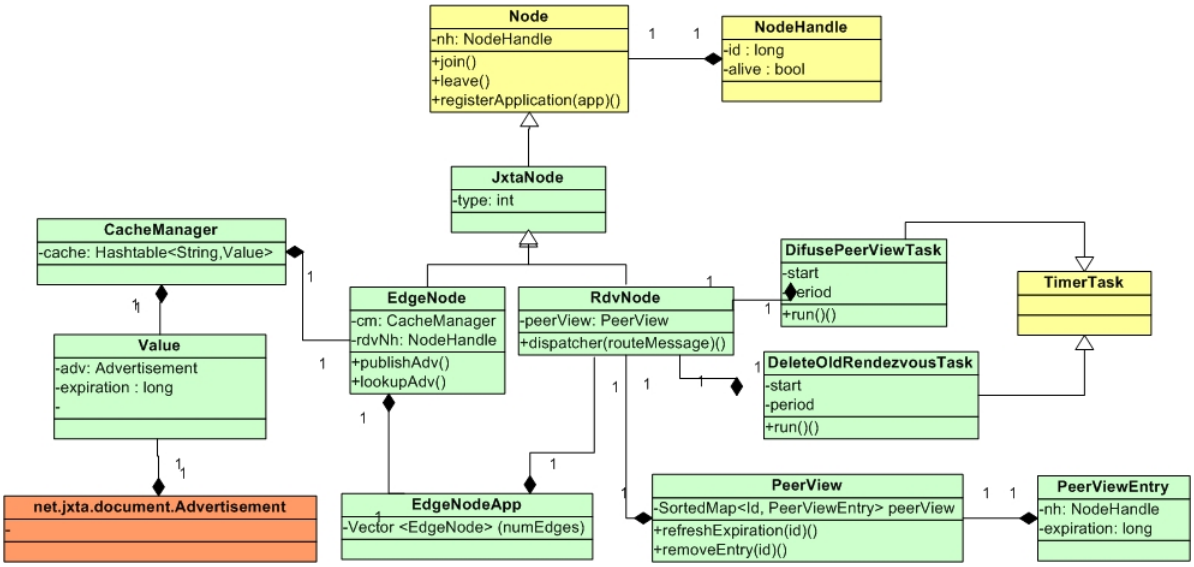


Figure 5.2: Classes in `planet.jxta` package

Figure 5.2 shows a class diagram of the main classes inside the `planet.jxta` package.

Jxta Nodes are the main components that implement the lookup algorithm. While edge peers publish and lookup advertisements, rendezvous nodes are responsible for the routing of messages and therefore are the ones that will send and receive more traffic.

The current version of PlanetSim allows to define just one type of node for the network. For this reason, a JxtaNode is used as the generic Node that extends from `planetsim.commonapi.Node` and `EdgeNode` and `RdvNode` extend from `JxtaNode`.

5.2.1 The `RdvNode` object

The `RdvNode` uses the information on its peer view to propagate messages. The package `rpv` includes the implementation of the peer view which is a Java `SortedMap` of rendezvous identifiers.

Nodes in PlanetSim can have different applications registered and that they can execute during the simulation. In JXTA-Sim an application called the `EdgeNodeManager` has been added to the `RdvNode` to manage edge peers. This class allows rendezvous node to easily manage the edge peers connected to them.

Another important element in the RdvNode is the *dispatcher*. The dispatcher is called at every step of the simulator and its function is to dispatch and handle route messages from the node incoming queue. Currently this element is implemented as a method but it could be refactored and implemented as another object.

Finally, two other classes are also related to the `RdvNode`: the `DifusePeerViewTask` and the `DeleteOldRendezvousTask`. These tasks are executed periodically and allow to refresh the entries of the peer view. The periodicity of their execution can be specified in the JXTA-Sim configuration file.

5.2.2 The `EdgeNode` object

The `EdgeNode` is simpler than the `RdvNode`. It does not have any tasks associated. The two main methods of this object are the `publish()` and the `lookup()` methods. Edge peers have a cache manager where published and discovered advertisements are stored. The cache manager is included in the package `cm` and it is implemented with a hashtable where the key is the identifier of the advertisement and the value is the advertisement.

Advertisements are XML documents. Advertisements in JXTA-Sim are created using the JXTA API. Therefore the advertisements stored in the cache of edge peers are instances of `net.jxta.document.Advertisement`.

Finally, edge nodes also have a dispatcher to handle incoming messages. The dispatcher will manage incoming requests and perform an action accordingly. Currently, edge peers communications in JXTA-Sim are limited to PUBLISH_ADV, LOOKUP_ADV and requests

5.3 Jxta Messages

During the simulation of the lookup algorithm nodes will send different messages to each other. When a node sends a message, it needs to specify the type of message so the receiver knows how to handle it. Additionally, PlanetSim allows to specify the mode of the communication since we could have the same type of message as a reply and a response.

There are three different modes of sending a message in Jxta-Sim:

REQUEST: Defines a communication that requires a response.

REPLY: Defines the response of a communication.

REFRESH: Defines a communication that is used for refreshing information about nodes or connections.

JXTA messages such as queries and responses, are described using XML documents. However, to reduce the processing load of the simulator, JXTA-Sim uses smaller and lighter messages for the communications.

Some types of messages will need additional information to be sent. For instance, the SEND_RPV reply sends a peer view to the destination. This information will be sent in an *application message* encapsulated within a *route message*.

5.3.1 JXTA-Sim application messages

JXTA-Sim uses four kind of application messages which are shown in figure 5.3.1.

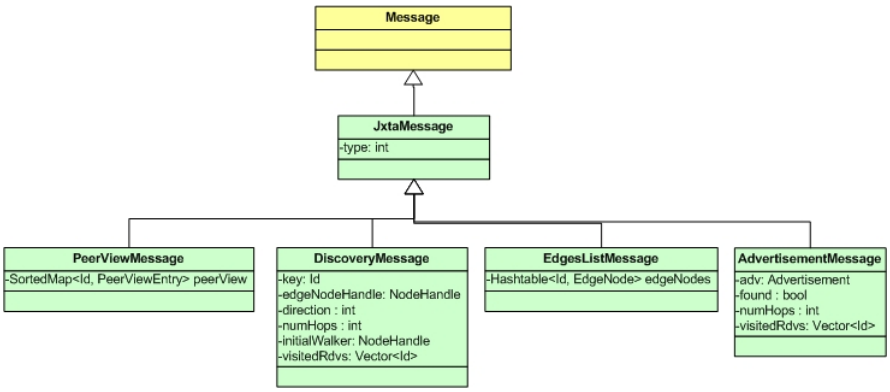


Figure 5.3: Application messages used in JXTA-Sim

Peer View Message: It is used to send a rendezvous peer view (RPV) to the destination of the message.

Edges List Message: It is used to send a list of edge nodes to the destination of the message.

Advertisement Message: It is used to send an advertisement to the destination of the message. This destination will be the edge peer requesting the advertisement. This message can also be sent to notify the requester that the advertisement was not found.

This message contains four fields:

found Boolean that indicates if the advertisement was found of not

advertisement The advertisement. If it was not found, then this field will be empty.

num hops Contains the number of hops necessary to find the advertisement.

visited nodes List that contains the nodes that were visited until finding the advertisement.

The last two fields are for gathering statistics at reception and its inclusion is optional.

Discovery Message: It is used to send a publish or lookup request. The value of the fields depends on the type of request (publish / lookup). In the case of a publication, the discovery message will be propagated in the rendezvous network until it has been replicated to the specified number of replicas. On the other hand, in the case of a search, it will be propagated until the advertisement is found or until the number of hops exceeds the maximum number of walker hops.

A discovery message is composed of the following fields:

key The key of the advertisement to be published/found

nodeHandle The node handle of the publisher or searcher. If the discovery message is used to publish a message, this field will correspond to the node handle of the publisher. The nodeHandle will be stored by the rendezvous storing the index of the advertisement, to be able to locate the owner when the advertisement is requested. If the discovery message is used for the lookup, it will correspond to the edge peer

searching for the advertisement. It will be used to route the advertisement from the edge that stores the advertisement to the searcher peer.

num hops The first time the discovery message is sent, this field will contain the maximum number of hops for this message. While traveling on the network, its value will be decreased by one. In the case of a publication, initially this will be the replication distance, while in the case of a search, this will be the maximum number of walker hops. Therefore, the discovery message will be propagated until the number of hops is zero.

direction This field is only used during the lookup. It indicates the direction in which the walker is traveling: up, down or none (initial value).

initial walker This corresponds to the handle of the rendezvous that starts the walker. This is used to avoid loops during the search. If the walk arrives at this rendezvous again, we must stop the search.

visited nodes List that contains the nodes that were visited until finding the advertisement. This field can not be used to reduce the message overhead.

5.3.2 JXTA-Sim message types

The following list shows the different types of messages used in Jxta-Sim and which simulate the communications during the JXTA discovery.

GET_RPV: This type of message is sent by a rendezvous peer to request a RPV to another rendezvous. It does not need to add additional information to the message.

SEND_RPV: This type of message is sent by a rendezvous peer as a response to the previous message. It uses a *Peer View Message* to send the peer view.

PUBLISH_INDEX : This message is generated by an edge peer to publish an advertisement to its local rendezvous. The edge peer pushes the index of the advertisement to the rendezvous by sending a *Discovery Message* which contains the necessary information to publish the advertisement. The local rendezvous computes a hash function to decide where the index will be located and forwards the discovery message to the chosen rendezvous.

STORE_INDEX : This message is sent by a rendezvous to propagate the index to other rendezvous after it has received a *Discovery Message* of type PUBLISH_INDEX. The rendezvous will forward the message with the *num hops* variable equal to the replication distance to the upper and lower rendezvous (successor and predecessor).

LOOKUP : This message is sent by an edge peer to its local rendezvous in order to find an advertisement. It uses a *Discovery Message* which contains the key of the advertisement to be found.

FORWARD_LOOKUP : This message is sent by a rendezvous peer after receiving a LOOKUP. It computes a hash function to choose the rendezvous where it will forward the lookup.

WALK_LOOKUP : This message is sent by a rendezvous peer after receiving a FORWARD_LOOKUP or WALK_UP message and not finding the index locally. In any case, it must continue (or start) the walker in the up and down directions.

RETURN_ADV : This message is sent by a rendezvous peer when the index of an advertisement has been found to the edge peer that owns the advertisement. This message is sent with a *Discovery Message* containing the source of the request.

SEND_ADV : This message is sent by an edge peer that owns an advertisement to the edge peer that looked up for it. The message sent is an *Advertisement Message* and contains the advertisement found.

PROBE_RDV, HELLO, CONNECT_TO_RDV, EDGES_CONNECTED : These messages are sent during the communications between a leaving rendezvous and another rendezvous when the first needs to transfer the edge peers to another active peer. These communications were shown in figure 4.3.4.

5.4 Running simulations

5.4.1 Configuring a simulation

The first step before running a simulation is to configure the simulation. JXTA-Sim allows the user to configure different parameters related to the lookup algorithm to evaluate its perfor-

Parameter	Description
MAX_EDGES_PER_RDV	Maximum number of edge peers per rendezvous
MAX_WALKER_HOPS	Maximum number of hops for the walker (in each direction)
MAX_PEERVIEW_SIZE	Maximum size of the Rendezvous Peer View to be sent to another node
REPLICATION_DISTANCE	Distance to replicate the index in the close rendezvous peers
CACHE_SIZE	Maximum size of the cache to store advertisements
RPV_ENTRY_DEFAULT_ EXPIRATION	Default expiration of an entry of in the RPV
PEERVIEW_REFRESH_ INTERVAL	Interval of time to send the some of the RPV to random rendezvous
PEERVIEW_ENRIES_ FLUSH_INTERVAL	Interval of time to remove the entries of the RPV which have expired
SIMULATION_STEPS	Number of steps to be simulated
NUM_PUBLISHERS	Number of peers that will publish an advertisement
NUM_SEARCHERS	Number of peers that will search for one of the published advertisements
RDVS_IN	Number of rendezvous that will join the network during the simulation
RDVS_OUT	Number of rendezvous that will leave the network during the simulation

Table 5.1: JXTA-Sim configurable parameters

mance.

These parameters are loaded from a file specified in the PlanetSim's master file and loaded into the class `JxtaProperties` from the package `planet.jxta`.

Table 5.1 provides a list of the parameters that the user of JXTA-Sim can configure.

5.4.2 Running a simulation

Simulations can be run by two means: by using an event file or programatically.

If we choose to use an event file, then the simulator will simply load the file, parse the events and execute them at the scheduled time. We can create this files manually or using the class `EventCreator` to create a file with random events scheduled at random times. Using event files allows to have repeatable simulations.

Alternatively, the developer can use a class that creates to create different tests that schedule different events and then start the simulator.

The code in listing 5.1 shows a sample of a code to create a simulator and simulate events by using an events file.

Listing 5.1: Starting a Simulation

```
NetworkSimulator sim;

Vector events = EventParser.parseEvents(Properties.simulatorEventFile);
Scheduler timer = new Scheduler();
timer.addEvents(events);

sim = new NetworkSimulator(timer);

for(int i=0; i<events.size();i++)
{
        sim.simulate();
}

addEdgeNodesToAllRdvs(sim.getInternalNetwork());

simulateRandomEvents();

processResults();
```

To run a simulation the first step is to create an instance of a `planet.simulate.NetworkSimulator`. Then, the different types of events can be added to the simulator by using a vector to store events. To simulate a step we can invoke the function `simulate()` from the object `NetworkSimulator`. This function calls the method `process()` of each of the Jxta nodes in the network.

5.5 Gathering Results

JXTA-Sim provides the user with different ways of gathering results:

Creating a graph of the network Using PlanetSim's results generator, JXTA-Sim allows to create a graph of the network at any desired time. Currently, JXTA-Sim gives this possibility at the end of the simulation but the user could write a class that allows to obtain this at other instants of time. This allows to see how the network varies during the simulation.

Figure 5.5 shows a graph of a network with 100 rendezvous nodes and 123 edge nodes. The network in the middle represents the rendezvous group and the green links are the connections with the edge nodes.

Gathering statistics during the simulation PlanetSim allows to to gather statistics using

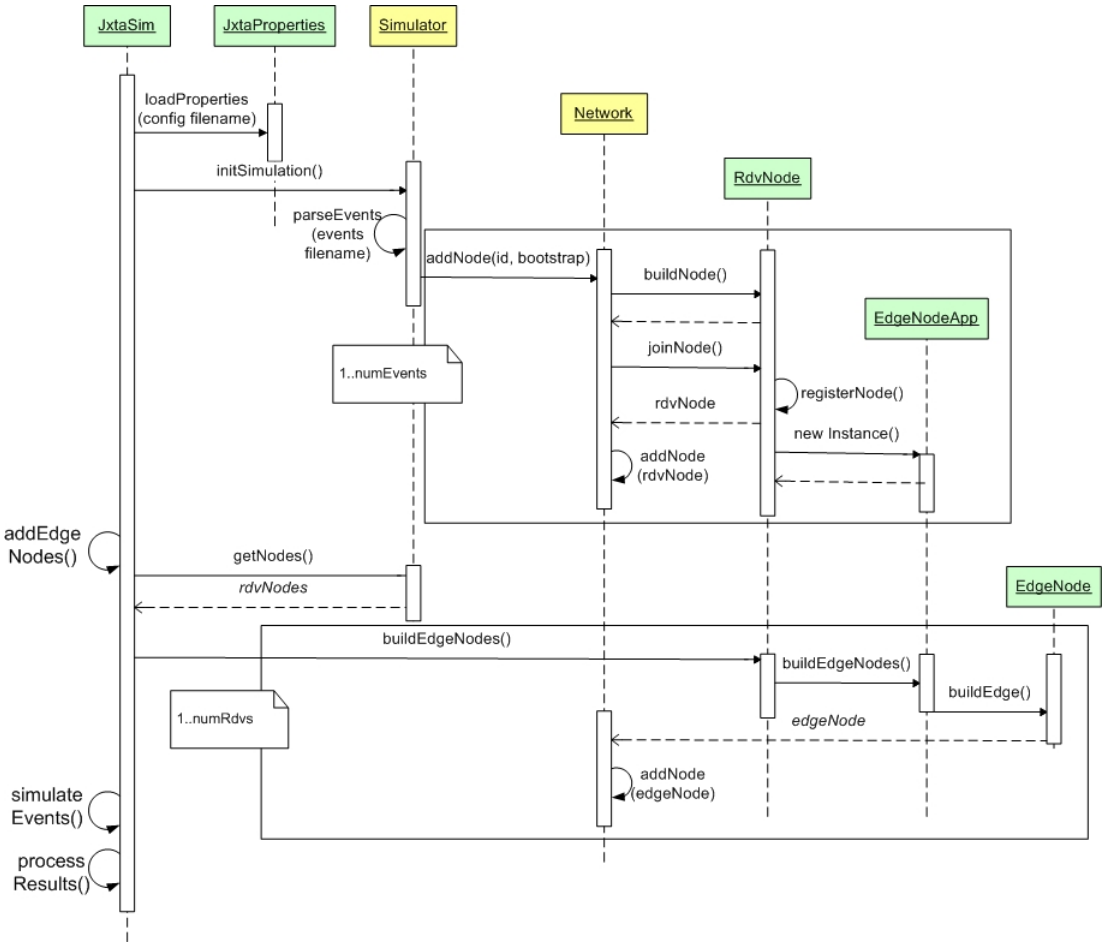


Figure 5.4: Sequence diagram showing the initialization of a simulation

Aspect Oriented Programming (AOP) and write them in a Gnuplot format. Gnuplot is a plotting utility that allows to generate 2D and 3D graphs. Currently PlanetSim can gather statistics about the different kind of traffic that is sent during the simulation.

JXTA-Sim introduces new aspects to generate graphs that show the different types of messages sent at every step of the simulation and the number of hops. Furthermore, other statistics could be easily generated by using aspects and gathering results during the simulation.

Figure 5.5 illustrates the number of hops for 100 lookups performed during 500 steps on a network with 500 nodes.

Gathering statistics at the end of the simulation Aspects can be useful to gather information during the simulation. However, users could also gather statistics at the end of the

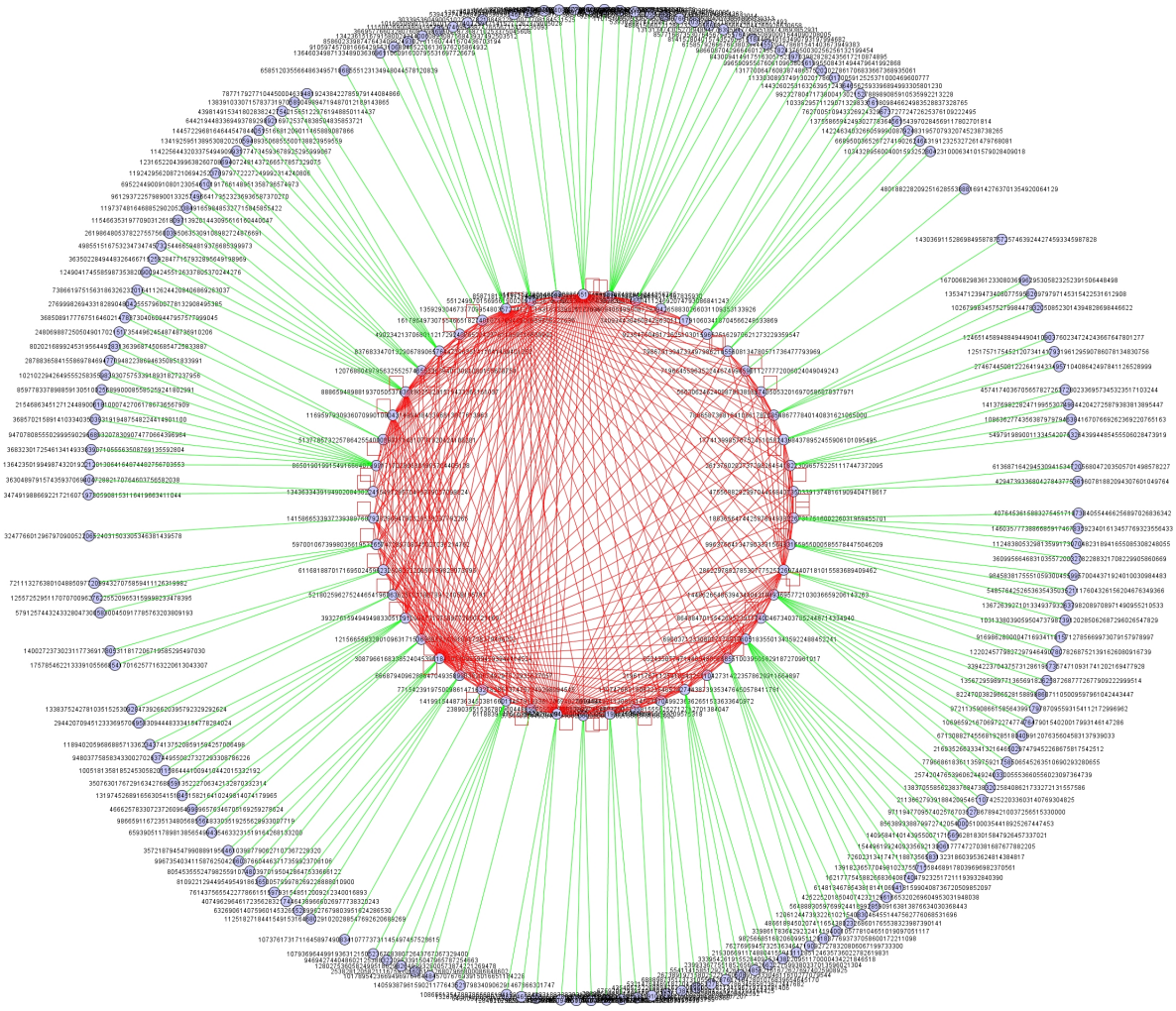


Figure 5.5: A JXTA network composed of 223 nodes

simulation.

For example, JXTA-Sim allows the user to obtain information about the connections between peers at the end of the simulation. Currently, the user can see the peer view of a rendezvous and the edge peers connected to a certain rendezvous.

Also, current tests try to calculate the percentage of found advertisements over a certain number of published advertisements and under the combination of different parameters. This is explained in more detail in section 6.2.

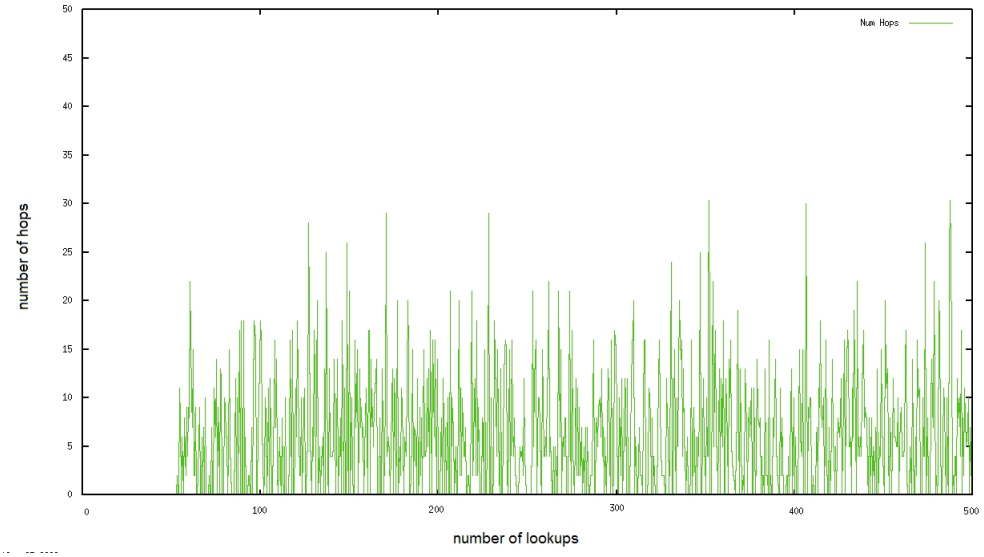


Figure 5.6: Number of hops per lookup on 100 lookups

Chapter 6

JXTA-Sim: Results and Evaluation

6.1 Evaluation of JXTA-Sim

After implementing JXTA-Sim tests that evaluate the correctness of the simulator must be done. To test if JXTA-Sim behaves as the the lookup algorithm, traces and graphs were generated during the simulation of the algorithm in different scenarios.

Doing tests in small networks and generating graphs that show the traffic of the network how the behaviour of the JXTA-Sim matches with the description of the JXTA lookup algorithm described in 2.4.

The graph in figure6.1 shows the types of messages sent during 60 steps in a simple network with 5 nodes. We have configured the simulation so one of the nodes leaves at step 28. By looking at this graph we can make the following observations which match the algorithm description:

1. Steps 0 to 28 show the nodes joining and requesting peer views to other nodes. The first node uses itself as bootstrap so it does not send any message to the network.

2. Steps 29 to 32 show the messages sent when a rendezvous decides to leave the network. The first three messages correspond to PROBE message sent to random rendezvous to check if they are active. All the probed nodes reply by sending a HELLO message each at step 30. The following step is to sent a message CONNECT_TO_RDV containing the list of edge peers to the first node that replied (the rest of HELLO messages are ignored). The last step is to send an acknowledgement confirming that the edge peers have been connected in an EDGES_CONNECTED message.

3. Steps 34 to 36 show the publication of an advertisement. The first message is sent by an edge peer that sends a PUBLISH request to its rendezvous. After a rendezvous receives a PUBLISH_INDEX request, it saves the index of the advertisement locally and computes a hash function using the key of the advertisement to choose where to store a replica. The second message correspond to a STORE_INDEX message sent from the first rendezvous to the chosen one. Finally, because in this example the REPLICATION_DISTANCE is equals to 1, the index is also stored in the upper and lower peers of the chosen one. Step 36 shows the two STORE_INDEX messages forwarded by the last rendezvous to the mentioned peers.

4. Finally steps 53 to 55 show the discovery of the previously published advertisement. The first one corresponds to the LOOKUP request message sent from the edge peer to its local rendezvous. Because the message is found on that peer the rendezvous sends a RETURN_ADV request

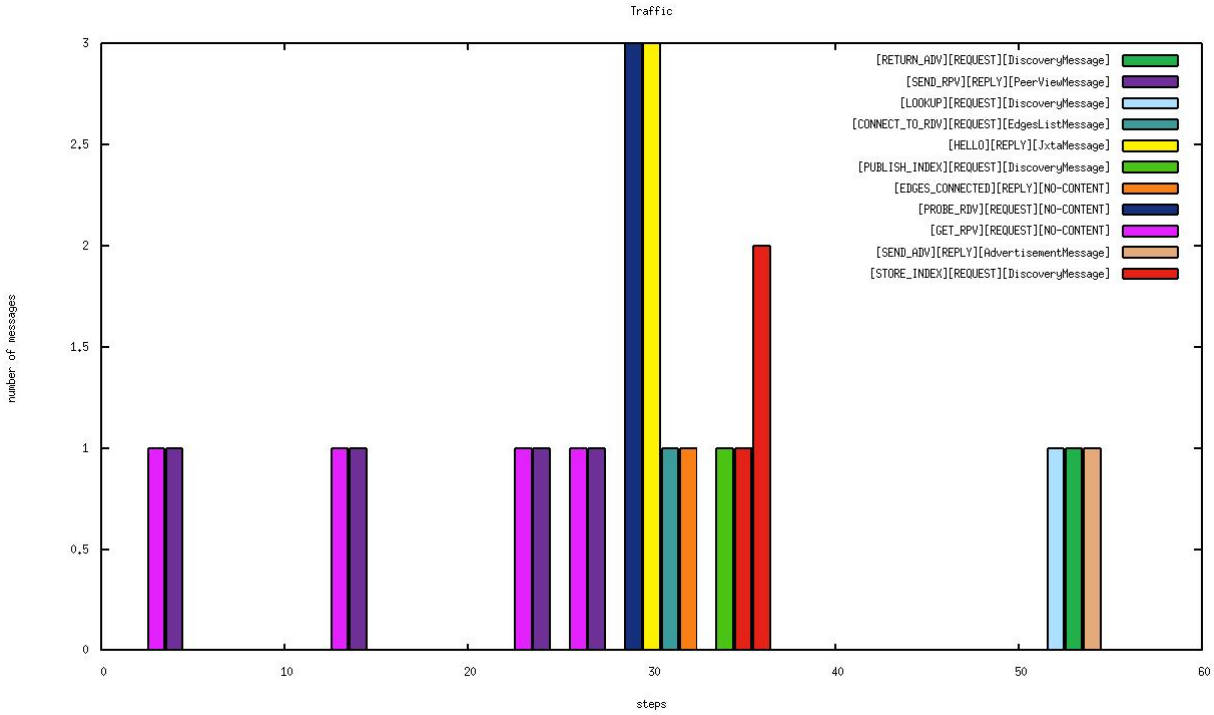


Figure 6.1: Messages sent during a simulation with 5 rendezvous nodes

We can also filter the traffic to evaluate a certain aspect of the algorithm. For instance, we can filter the messages related to the maintainance of the network, that is, the messages sent

by the rendezvous node tasks. Figure 6.1 show the messages sent by rendezvous to probe other rendezvous by sending a portion of their peer views. The interval between probings is set to 200 steps and they all start the tasks at interval (200 + rn) where *rn* represents a random value, so that they do no send these messages at the same times. The number of rendezvous to send the peer view is also random.

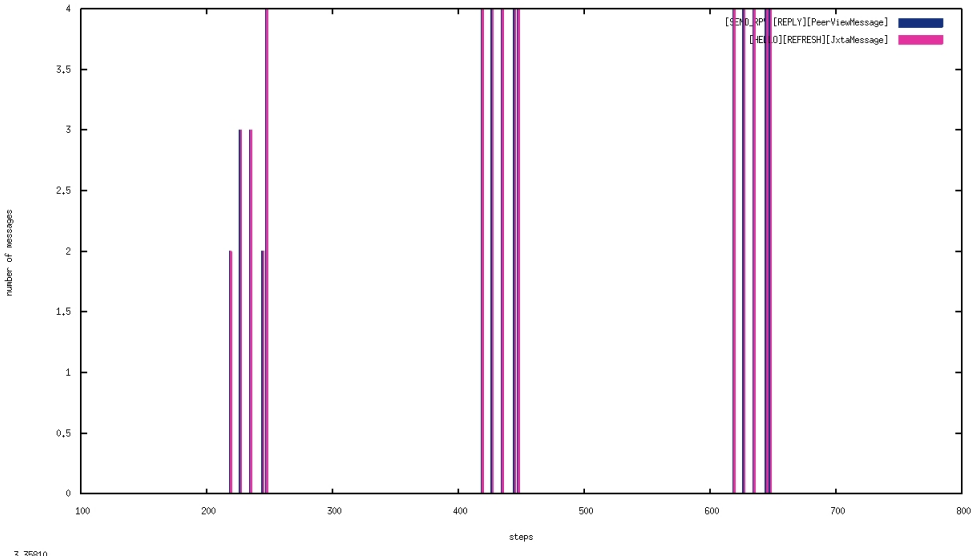


Figure 6.2: Maintentance messages sent during a simulation with 5 rendezvous nodes

Traces have also been useful to study the messages sent and the events occurring in the simulator. For instance, listing 6.1 shows a trace after a rendezvous that gets disconnected from the network with 5 rendezvous nodes. The trace shows the different kind of messages involved in the communication between the leaving rendezvous and other rendezvous that will try to offer connectivity to the edge peers of the leaving peer. Each trace includes the identifier of the peer sending or receiving the message.

Listing 6.1: Trace of the disconnection of a Rdv

```
1  Rdv {13204415936568644089364746615992621000867511628 35} leaving the network...
2
3  ----------
4  Edges connected to {13204415936568644089364746615992621000867511628 35}
5
6  14026041825263173303030231804589578623034749994 17
7  20855557160319006796851629707378693308427036234 5
8  64973898149372508211333721880269126097789939909 3
9  ----------
10
11 {13204415936568644089364746615992621000867511628 35} Sending Probe Message
12  to rdv{ 71090586433683702908525579921071190811153636027}
13 {13204415936568644089364746615992621000867511628 35} Sending Probe Message
14  to rdv{ 90700122045964915718525576100456087946757628 5657}
15
16 {71090586433683702908525579921071190811153636027}PROBE_RDV
17 Message Received!!
18 {90700122045964915718525576100456087946757628 5657}PROBE_RDV
19 Message Received!!
20
21 {13204415936568644089364746615992621000867511628 35} HELLO
22 Message received!
23 {13204415936568644089364746615992621000867511628 35} HELLO
24 Message received!
25
26 {71090586433683702908525579921071190811153636027}CONNECT_TO_RDV
27 Message Received!! Peers added!
28
29 ----------
30 Edges connected to {71090586433683702908525579921071190811153636027}
31
32 10889954140133623708678065151043284181390263220 33
33 12484312987399798045119800737160435763892515791 7
34 14026041825263173303030231804589578623034749994 17
35 20855557160319006796851629707378693308427036234 5
36 28106874604087028636695060128823150788337807624 9
37 64973898149372508211333721880269126097789939909 3
38 ----------
39
40 {13204415936568644089364746615992621000867511628 35}
41 EDGES_CONNECTED Msg Received! New edges connected successfully to Rdv!
```

Line 2 shows the intention to leave of the peer with identifier {*132...835*}. Lines 8-10 show the edge peers connected to the peer that is about to leave. Lines 13-16 show the two probe messages sent to two random rendezvous. We can see how the leaving peer tries to connect to peer {*710...027*} which is the peer from whom it received the first *hello* message. Lines 32-39 show the new edge peers added to the new rendezvous. Finally, line 43 shows the leaving rendezvous receiving a message confirming that edges were added. After this message, the peer leaves the network.

Another option to study the network is to generate network graphs during different times of the simulation. Figures 6.1 and 6.1 show a graph of the network with 5 initial rendezvous nodes before and after node {*180...646*} leaves. The latest figure show how its edge nodes are migrated to peer with identifier {*710...027*}.

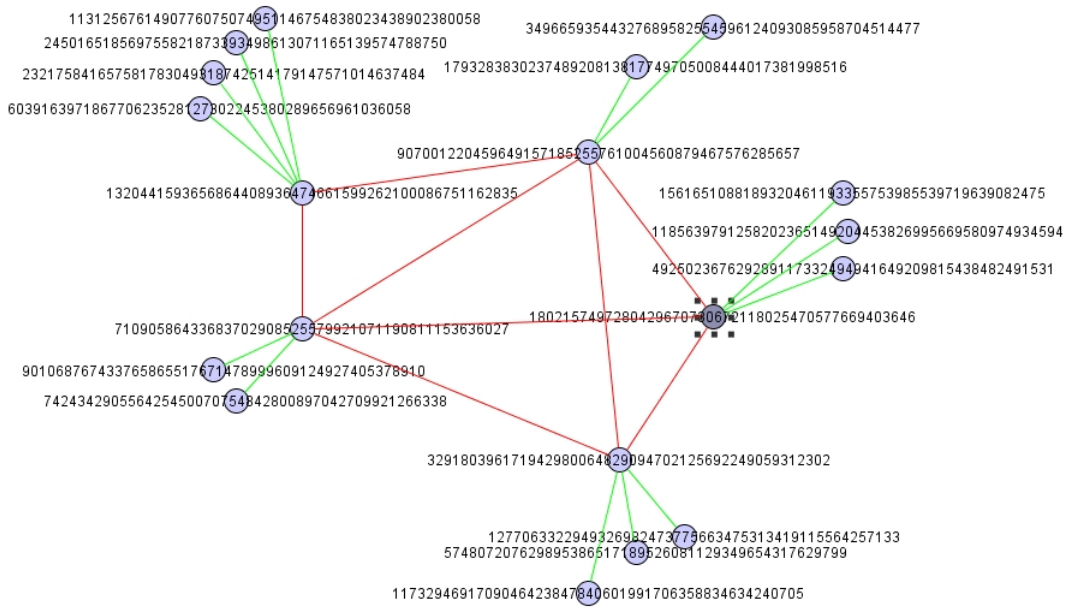


Figure 6.3: Network with 5 rendezvous nodes before one of them leaves

By performing different tests and studying the traces and graphs, it has been proven that JXTA-Sim behaves as it should.

6.2 Evaluation of the JXTA Lookup Algorithm

After evaluating the correctness and accuracy of the simulator, another key point for the evaluation of JXTA-Sim is to perform tests to evaluate the JXTA lookup algorithm. Different tests

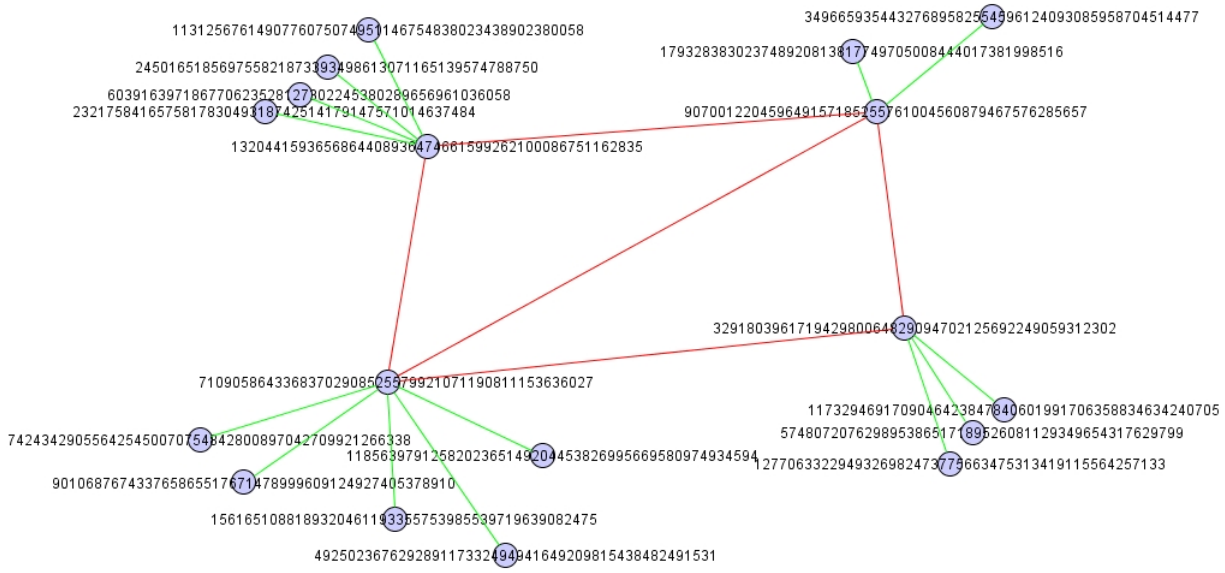


Figure 6.4: Network with 5 rendezvous nodes after one of them has left

can be done by changing the parameters related to the algorithm seen in table 5.1.

To do the tests that will evaluate the JXTA lookup algorithm the metrics chosen have been the percentage of successful searches and the number of hops needed to find an advertisement.

By studying the number of hops we can evaluate the efficiency of the distributed hash table (*how many of the advertisements can found by only using the DHT?*) and of the walker (*how many hops does the walker need to do to find an advertisement?*).

Experiments have shown that with a small number of nodes, the number of successful searches are very high. More importantly, the number of walker hops doesn't need to be very high and for most of the searches the number of hops is equals to two, which means that only the DHT was used. In a network with 200 nodes, 50 of which are rendezvous and 150 are edge nodes, with replication distance equals to 1, a maximum number of hops equals to 5 and no churn, up to 95% of the advertisements could be found.

When churn is added in this scenario, the number of successful searches decreases but by increasing the replication distance by 1 or 2, we can get better results. Figure 6.2 shows in blue the percentage of successful searches when we add a churn of 20% (this is, a 20% of the rendezvous leave and another 20% will join), which decreases about 10% with respect to the scenario without churn. We can observe that, by increasing the replication distance by 2, the

number of successful searches is increased again and we can find up to 98% of the advertisements when we perform 10 queries.

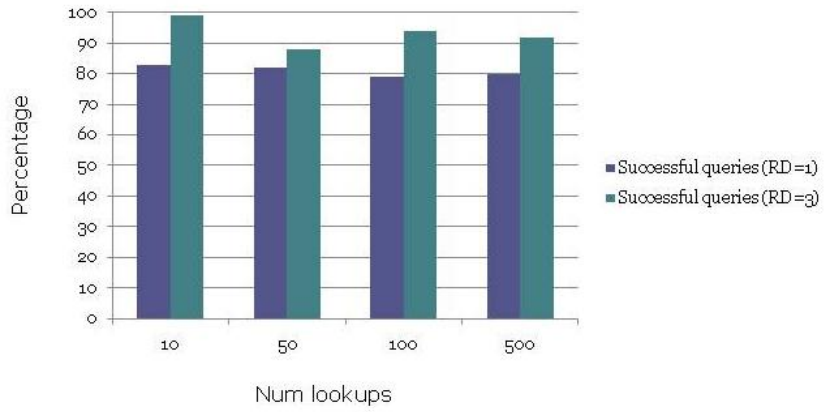


Figure 6.5: Percentage of successful searches in a network with 50 rendezvous nodes, a maximum number of hops equal to 5 and a churn of 20% under different replication distances

As we increase the number of nodes and add churn, the percentage of successful searches decreases. However, increasing the replication distance by 1 or 2 considerable improves the results. For instance, in a network with 4000 nodes, 1000 of which are rendezvous nodes, a churn of 10% and a replication distance of 1, the percentage of successful searches is less than 50% which shows a very poor performance. However, as we increase the replication distance, results increase. With a replication distance of 4, the percentage goes up to 73% (see figure 6.2). More experiments could not be done because of time limitations but we expect the results to improve as we increase the replication distance. More study in this line is left as future work.

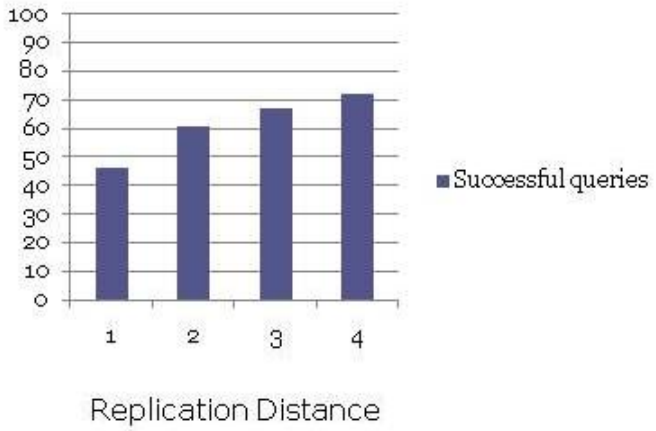


Figure 6.6: Percentage of successful queries for different *Max Walker Hops* values

By looking at this experiments we can argue that although performance decreases in scenarios with a high churn, increasing the *replication distance* helps to compensate the effect of the churn.

In the same scenario, if we try to improve the results by increasing the maximum number of walker hops and leaving the replication distance fixed to 1, we need to double the number of hops to achieve the same improvement as with a replication distance of 4 (see figure 6.2). Doubling the number of hops would mean that the walker could perform up to 20 hops which would increase considerably the network traffic. Therefore, the recommended option is to increase the replication distance.

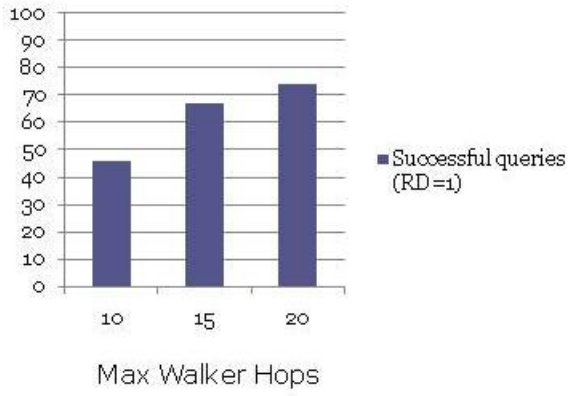


Figure 6.7: Percentage of successful queries for different *Replication Distance* values

We believe that better results could be obtained by the combination of the two parameters, which is also left as future work.

These experiments show that being able to change these and other parameters helps to adapt the algorithm to the current scenario. Currently most of this parameters are given by default in JXTA. However, these experiments prove that the JXTA platform should allow the developer to modify this parameters to deal with different situations (e.g. different network sizes and different churns).

These experiments have been performed by modifying the size of the network, the number of hops, the replication distance and the number of lookups. Other interesting conclusions could be observed by doing tests which modify other parameters. A discussion about this can be found in section 7.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The goal of this project has been to build a simulator for the JXTA lookup algorithm. The simulator allows to study and evaluate the algorithm by modifying different parameters. Moreover, JXTA-Sim satisfies the initial requirements:

Scalability: The performed experiments have allowed to test up to 10.000 nodes. However, experiments with a larger number of nodes could be performed to evaluate the scalability.

Extensibility: JXTA-Sim has been implemented following an object oriented approach which allows to easily extend the simulator to evaluate other aspects of JXTA. Also the JXTA-Sim design tries to be simple and the code is well commented to make it easier to understand.

Usability and Flexibility: JXTA-Sim allows to evaluate the JXTA lookup algorithm by performing different experiments with different parameters. Users can obtain different types of results such as graphs and statistics of the simulated scenarios and they can also code new tests to obtain other results.

By scanning the traffic of JXTA-Sim using traces and graphs we can prove that the algorithm behaves as it is described in section 2.4.

This project has also shown how to study some of the aspects of the algorithm by using JXTA-Sim. Furthermore, experiments with the simulator show that the algorithm scales quite well to the number of nodes and that it does not need a large number of hops to find most of

the advertisements. In fact, up to 95% of the advertisements can be found on a network of 50 rendezvous nodes with a maximum number of hops equals to 5. The reason to this could be because the DHT function used to calculate the node to look for the index makes the walker walk towards the node whose identifier is closer to the index, which optimises the search.

Other observations that can be taken from the experiments are that although performance decreases in scenarios with a high churn, increasing the *replication distance* helps to improve it. Usually increasing it to 1 or 2 more steps shows much better results. On the other hand, to obtain the same improvement by increasing the *maximum number of walker hops* requires a much higher increase in the number, which would add more traffic to the network.

In any case, being able to change these and other parameters helps to adapt the algorithm to the current scenario. Currently most of this parameters are given by default in JXTA. However, these experiments prove that the JXTA platform should allow the developer to modify this parameters to deal with different situations (e.g. different network sizes and different churns).

7.2 Future Work

This project has shown how to build a simulator to simulate and evaluate JXTA's lookup algorithm. There are different lines of study that can be followed from here.

One line of study would be to add improvements to the simulator and to extend its functionality to simulate other parts of JXTA. JXTA-Sim is implemented using an object oriented approach which allows to define a simple architecture where components (Java objects) can be easily extended.

As seen in section 4.1.2, some assumptions have been taken into account to simplify the design of JXTA-Sim. These assumptions lead to simulations of ideal scenarios where nodes don't fail and firewalls or NAT don't exist. Jxta-Sim can be extended to include more realistic scenarios. PlanetSim developers have been working on an extension that introduces latencies in the network. Furthermore, their network layer could be integrated with an existing network simulator such as NS2 or OMNET++ to achieve more realistic networks.

The current version of JXTA-Sim does not reproduce the probing of edge peers. In JXTA, edge peers periodically probe their rendezvous to check if they are active and if they aren't edge peers need to find other rendezvous or promote themselves to rendezvous peer if they do

not find any. In JXTA-Sim, if a rendezvous peer connected to an edge peer leaves the network it will "'migrate"' their edge peers to another active rendezvous. This is done to improve the performance of the simulator, especially in situations with a large number of peers. However, researchers may want to reproduce this aspect to obtain more accurate simulations. Although this could be done easily, adding many tasks on the nodes could slow down the simulator performance and is not recommended in scenarios with many nodes. The ideal scenario would be that the user of the simulator could choose the option.

To be able to transfer edge peers from one rendezvous to another, a rendezvous peers needs to indicate that is leaving instead of failing and disconnecting suddenly. For this reason, one assumption that was taken was that rendezvous peers never fail. Another assumption was that edge peers can not leave the group. This assumption was done to be able to study the percentage of successful searches over a certain number of published advertisement with the limited walker as the only responsible of a failed search. These results then, consider that peers do not fail. Researchers may want results which are more realistic. For this reason, the simulator could be extended to include the failure of nodes and allow the user to indicate if peers will fail and how many.

Another assumption of JXTA-Sim is that there is a unique peer group and all peers belong to this group making all the published advertisements visible to everyone. It would be interesting to include the concept of peergroups and the possibility of performing experiments with different groups (different search scopes) where peers could belong to multiple groups.

Currently JXTA-Sim only simulates the behaviour of the JXTA Discovery and Routing protocols. However, future versions of JXTA-Sim could also simulate other protocols such as the Pipe Binding Protocol which implements virtual communication channels (pipes) that can be used to send messages between peers. The PlanetSim network could be extended to include the concept of pipes to evaluate their performance which until now has only been evaluated using testbeds [Halepovic and Deters, 2005], [Halepovic and Deters, 2003a]. It could also be interesting to integrate the Peer Endpoint Protocol which is the protocol which is used by peers when there is no direct route between them (e.g. there could be a firewall in between). The endpoint protocol introduces the concept of relay peers which are contacted by peers when they need to find routes when other peers are unreachable or temporarily unavailable.

Another line of study would involve performing more experiments with JXTA-Sim to obtain more results about the algorithm. Some experiments have been made by modifying the replication distance and the number of hops. However it would also be interesting to study how the algorithm is improved by modifying other parameters. For instance, we could evaluate how the size of the peerview changes the network size and the churn are increased. Other parameters that could be modified are the periodicity of the tasks to refresh entries and probe other rendezvous. In scenarios with a lot of traffic it may be better to have a high interval while in scenarios with high mobility a smaller value would be more suitable.

JXTA claims to be suitable for mobile ad hoc networks where peers join and leave the network frequently. Another line of study using JXTA-Sim is to compare the JXTA lookup algorithm to other overlay approaches such as Chord (which is currently implemented in PlanetSim), Pastry or Gnutella and evaluate the improvement (or decrease) of JXTA in situations with high churn and unstable connections.

Finally, one of the possibilities of JXTA-Sim is to simulate an application running on top of the JXTA lookup algorithm. PlanetSim supports the registration of applications to nodes and currently they have implemented a simulation of Scribe over Chord. Other applications, such as mobile applications could be simulated using JXTA as the overlay network to test their performance using JXTA routing.

Bibliography

[omn, 2009] (2009). WWW page. www.omnetpp.org/.

[yed, 2009] (2009). WWW page. www.yworks.com/products/yed/.

[fre, 2009] (2009). Freepastry. www.freepastry.org/FreePastry/.

[gml, 2009] (2009). Geography markup language. WWW page. http://www.opengeospatial.org/standards/gml.

[gnu, 2009a] (2009a). Gnuplot. WWW page. http://www.gnuplot.info/.

[gnu, 2009b] (2009b). Gnutella website. WWW page. www.gnutella.com.

[paj, 2009] (2009). Pajek wiki. WWW page. http://pajek.imfm.si/doku.php.

[Ahkil et al., 2003] Ahkil, B. T., Traversat, B., Arora, A., Abdelaziz, M., Duigou, M., Haywood, C., christophe Hugly, J., Pouyoul, E., and Yeager, B. (2003). Project jxta 2.0 super-peer virtual network. Technical report, Sun Microsystems, Inc.

[Antoniu et al., 2007] Antoniu, G., Cudennec, L., Jan, M., and Duigou, M. (2007). Performance scalability of the jxta p2p framework. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10.

[Baumgart et al., 2007] Baumgart, I., Heep, B., and Krause, S. (2007). Oversim: A flexible overlay network simulation framework. In *IEEE Global Internet Symposium, 2007*, pages 79–84.

[Belmonte et al., 2007] Belmonte, M. V., Diaz, M., Perez-de-la Cruz, J. L., and Reyna, A. (2007). File sharing service over a generic p2p simulator. In *SKG '07: Proceedings of the*

Third International Conference on Semantics, Knowledge and Grid, Washington, DC, USA. IEEE Computer Society.

[Brown and Kolberg, 2006] Brown, A. and Kolberg, M. (2006). Tools for peer-to-peer network simulation. Technical report, Internet Draft:draft-irtf-p2prg-core-simulators-00.

[Clarke et al., 2001] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. (2001). Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*.

[E. Halepovic, 2004] E. Halepovic, R. Deters, B. T. (2004). Performance evaluation of jxta rendezvous. In *International Symposium on Distributed Objects and Applications (DOA '04)*, Agia Napa, Cyprus. Springer Verlag.

[Erich(extern), 2003] Erich(extern), S. (2003). The neurogrid project. In *International Conference on Computational Intelligence for Modelling, Control and Automation - CIMCA'2003*, Vienna, Austria.

[Gradecki, 2002] Gradecki, J. D. (2002). *Mastering JXTA*. Wiley, Indianapolis Ind.

[Halepovic and Deters, 2003a] Halepovic, E. and Deters, R. (2003a). The costs of using jxta. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, Washington, DC, USA. IEEE Computer Society.

[Halepovic and Deters, 2003b] Halepovic, E. and Deters, R. (2003b). Jxta performance study. In *Communications, Computers and signal Processing, 2003. PACRIM. 2003 IEEE Pacific Rim Conference on*, volume 1, pages 149–154 vol.1.

[Halepovic and Deters, 2005] Halepovic, E. and Deters, R. (2005). The jxta performance model and evaluation. *Future Gener. Comput. Syst.*, 21(3):377–390.

[Kotilainen et al., 2006] Kotilainen, N., Vapa, M., Keltanen, T., Auvinen, A., and Vuori, J. (2006). P2prealm peer-to-peer network simulator, in. In *Proc. 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks, 2006*. Unpublished.

[Mohamed et al., 2003] Mohamed, B. T., Abdelaziz, M., and Pouyoul, E. (2003). Project jxta: A loosely-consistent dht rendezvous walker.

[Naicken et al., 2007] Naicken, S., Livingston, B., Basu, A., Rodhetbhai, S., Wakeman, I., and Chalmers, D. (2007). The state of peer-to-peer simulators and simulations. *SIGCOMM Comput. Commun. Rev.*, 37:95–98.

[Pujol-Ahulló et al., 2009] Pujol-Ahulló, J., García-López, P., Sànchez-Artigas, M., and Arrufat-Arias, M. (2009). An extensible simulation tool for overlay networks and services. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2072–2076, New York, NY, USA. ACM.

[Schlosser et al., 2002] Schlosser, M. T., Condie, T. E., Kamvar, S. D., and Kamvar, A. D. (2002). Simulating a p2p file-sharing network. In *in First Workshop on Semantics in P2P and Grid Computing*.

[Schollmeier, 2001] Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *P2P '01: Proceedings of the First International Conference on Peer-to-Peer Computing*, page 101, Washington, DC, USA. IEEE Computer Society.

[Theodoloz, 2004] Theodoloz, N. (2004). Dht-based routing and discovery in jxta. Master's thesis, Ecole Polytechnique Federale de Lausanne.

[Traversat et al., 2002] Traversat, B., Abdelaziz, M., Duigou, M., Hugly, J. C., Pouyoul, E., and Yeager, B. (2002). Project JXTA virtual network.

[Wei et al., 2007] Wei, G., Chunhe, X., Nan, L., Haiquan, W., and in, D. (2007). Research on simulation framework of structured p2p network. In *FGCN '07: Proceedings of the Future Generation Communication and Networking*, Washington, DC, USA. IEEE Computer Society.