Performance Evaluation and Benchmarking of the JXTA Peer-To-Peer Platform

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of Master of Science

in the Department of Computer Science

University of Saskatchewan

Saskatoon

By

Emir Halepovic

Keywords: benchmark, JXTA, peer-to-peer, performance

Permission to Use

Abstract

Peer-to-peer (P2P) systems are a relatively new addition to the large area of distributed computer systems. The emphasis on sharing resources, self-organization and use of discovery mechanisms sets the P2P systems apart from other forms of distributed computing.

Project JXTA is the first P2P application development platform, consisting of standard protocols, programming tools and multi-language implementations. A JXTA peer network is a complex overlay, constructed on top of the physical network, with its own identification scheme and routing.

This thesis investigates the performance of JXTA using benchmarking. The presented work includes the development of the JXTA Performance Model and Benchmark Suite, as well as the collection and analysis of the performance results. By evaluating three major versions of the protocol implementations in a variety of configurations, the performance characteristics, limitations, bottlenecks and trade-offs are observed and discussed.

It is shown that the complexity of JXTA allows many factors to affect its performance and that several JXTA components exhibit unintuitive and unexpected behavior. However, the results also reveal the ways to maximize the performance of the deployed and newly designed systems.

The evolution of JXTA through several versions shows some notable improvements, especially in search and discovery models and added messaging components, which make JXTA a promising member of the future generation of computer systems.

Acknowledgements

I would like to thank my supervisor Dr. Ralph Deters for his guidance, advice and constant belief that I will do the right thing throughout this research. His outstanding assistance in achieving the publications and support to attend the conferences are so much appreciated.

The comments of the advisory committee have helped make this thesis better and I would like to thank all members.

I thank the members of JXTA community for their efforts, cooperation and enthusiasm in producing this one-of-a-kind open-source software platform.

With great pleasure and pride, I would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC), for their sponsorship in pursuing this graduate program.

By making sure that no deadline is missed or detail forgotten, Ms. Jan Thompson has made this journey so cheerful and worry free.

Finally, this thesis would be impossible to complete without my absolutely irresistible girlfriend Maja. Her enormous love, support, tolerance and understanding (of me, not JXTA) made me a better person, student and hopefully IT professional; and she managed all that while pursuing her own degree in Nutrition.

E. H.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
DNS	Domain Name System
DHT	Distributed Hash Table
GB	Gigabyte
GHz	Gigahertz
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISP	Internet Service Provider
JXTA	From “juxtapose” – a peer-to-peer platform
JVM	Java Virtual Machine
KB	Kilobyte
Kbps	Kilobits per second
LAN	Local Area Network
MB	Megabyte
Mbps	Megabits per second
MHz	Megahertz
NFS	Network File System
NAT	Network Address Translation
P2P	Peer-to-peer
RAM	Random Access Memory
SRDI	Shared Resource Distributed Index
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Universal Resource Locator
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
WAN	Wide Area Network
WSDL	Web Services Description Language
XML	Extensible Markup Language

Chapter 1

Introduction

Project JXTA is an open-source effort to formulate, develop and standardize the core peer-to-peer (P2P) operations and protocols. JXTA (pronounced “juxta”, from *juxtapose*) defines protocols for discovery, messaging, identification, group organization, etc., which are necessary for all P2P applications. JXTA aims to leverage and build upon the traditional communication protocols, such as TCP and HTTP, and to provide universal components for building P2P applications.

Over three years of development in an open-source community have produced a solid API, largely tested in academic environments as a platform for prototyping various P2P concepts. As the JXTA platform matures and commercial applications are appearing in the market, it is necessary to evaluate and characterize it. This thesis aims to provide a performance evaluation of the JXTA platform and its components using benchmarking.

1.1 Motivation and Goals

The motivation for this research is found in the lack of performance data about JXTA and the benefits that such data may offer to both the developer and research communities. Specifically, the platform developers will benefit from better understanding the behavior of JXTA components in various environments. The designers and users of P2P systems based on JXTA will know the performance limits of their products and have the guidelines to improve their designs and implementations. Finally, the P2P research based on simulation will be provided with realistic parameters obtained from real-life experiments.

This research has the following goals:

- ❖ Design and develop a JXTA Performance Model and Benchmark Suite.
- ❖ Obtain performance bounds and identify the bottlenecks of the components and systems built on the JXTA platform, based on the benchmarking results in a variety of network configurations.
- ❖ Formulate the deployment patterns that achieve the desired performance levels of the JXTA system.

1.2 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 presents an overview of P2P systems and their performance characteristics in general. Chapter 3 gives a background on JXTA P2P platform, protocols and previously established performance results. The goals and phases of the research are reviewed in Chapter 4. The process of developing the Performance Model and Benchmark Suite is described and discussed in Chapter 5, followed by the analysis of the performance results in Chapter 6. The lessons learned from this research are presented in Chapter 7 and summary and contributions conclude the thesis in Chapter 8.

Chapter 2

Peer-to-Peer Computing Overview

This chapter provides background information on P2P systems. Definition and types of P2P systems and architectures are presented. The goals, benefits and problems of P2P approach are briefly discussed, as well as the representative applications of all types of P2P systems. Different approaches to object location in P2P systems are explained and the current performance evaluation methods and results presented.

2.1 What is P2P

The P2P approach has gained significant success and popularity in recent years, due to several widespread types of application: file sharing, collaboration and distributed computing. Although a strict definition of P2P is yet to be established, it is widely accepted that P2P systems are characterized by decentralized control, high autonomy of participating nodes and heterogeneity in terms of processing power, hardware and software resources. Possibly the most detailed and precise definition of P2P is given by Clay Shirky [52]:

> *P2P is a class of applications that takes advantage of resources - storage, cycles, content, human presence - available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers.*

Traditional Internet computing is based on a client-server model, in which a resource-rich computer provides resources to clients. The clients are typically home or office computers that consume a variety of resources, such as web pages or an office database.

A “shift of power” occurs in P2P systems, where all participating nodes (peers) can act as both clients and servers. Each peer provides some resources to the peer community, according to its capabilities. At the same time, a peer consumes the resources of other peer(s). The resources are common for a specific peer community. In the most popular file-sharing communities, all kinds of files are exchanged between peers. The other peer communities provide the processing power or virtual office resources (email, whiteboard, file repository, etc.)

According to Milojicic et al. [31], P2P should not be confused or used as an interchangeable term with traditional distributed computing, grid computing or ad-hoc networking. Distributed computing is usually associated with computing clusters, in which several computers are interconnected and share an assigned computing task, whereas grid computing is a controlled and coordinated large scale sharing of computing resources at the multi-organizational level [31]. Ad-hoc networks refer to any system dynamically connected without prepared infrastructure, which represents only one aspect of P2P systems. P2P systems differ from computer clusters in their size and resources being shared and from grids mainly in the lack of organization and coordination between nodes.

2.2 What is a Peer

Peers are defined in an application-specific way. A peer that belongs to a particular P2P network may represent a physical computer or device, a single program or a set of programs running on a single computer or even a human user. Peers may communicate by connecting directly to each other, using routing nodes or even a central server. In case of human peers and instant messaging, the communication only makes sense if the peers know about each other. In other cases, it is just the common goal that the peers share, which gives a P2P character to the application, such as in SETI@home [50] or medical research applications, where peers may never connect to each other directly.

Computer nodes that participate in a P2P system are located at the “edge” of the Internet (e.g. users’ homes), with generally intermittent connectivity and changing identification (IP address) between sessions. Users of a system activate applications on a peer computer to accomplish common goals, such as share files [8, 25], collaborate [33] or participate in a world-wide distributed computation effort [50].

2.3 Goals, Benefits and Problems of P2P Systems

Although P2P is not a new concept, it does offer new solutions to both known and emerging problems. Usenet and Domain Name System (DNS) are similar to P2P in the aspect of collaboration of independent nodes to achieve the common goal. The Usenet system provides access to newsgroups (message boards) hosted worldwide by independent servers [23]. DNS servers translate the computer names to IP addresses allowing communication between any two computers connected to the Internet [32].

The most important benefits of P2P systems are as follows:

- ❖ **Use of the previously unused resources:** On home and office computers, processing cycles are wasted constantly while the computer is on but underutilized/idle (generally overnight and during non-business hours). The disk storage is typically underutilized, as these computers are used mostly for simple non-intensive tasks. The P2P-based application can make use of these resources, thereby increasing utilization of an already paid resource.
- ❖ **Potential to scale:** The resources of the server or server-cluster limit the capabilities of the client-server system. As the number of clients increases, it becomes very difficult to keep up with demand and maintain the performance and service at the required level. By distributing demand and load on the shared resources, the bottlenecks can be eliminated and a more reliable system achieved.
- ❖ **Ad-hoc connectivity and self-organization:** P2P systems build and organize themselves. Peers dynamically discover each other and build the network and they organize according to their preferences and current conditions within the

peer group. For example, P2P systems such as file sharing networks offer a choice of file providers for download. If a popular peer is overloaded and poor performance occurs, consumer peers can switch to another provider, effectively rebalancing load and changing the network topology.

- **Increased autonomy and anonymity:** In P2P systems, peers are all of equal status. They can autonomously decide when and for how long to participate and how much of their resources to share. Similarly, it is possible to achieve higher anonymity and privacy simply by having no central authority that can keep track of the activity in the system.
- **Cost distribution and reduction:** This benefit actually comes through the achievement of all other goals. The cost of a powerful server or cluster can be avoided by distributing processing tasks over numerous low-powered computers. By letting the system self-organize into a large peer group, there is no need for central administration and maintenance, which reduces cost as well.

However, P2P also raises many concerns, mainly about security, manageability and performance. Several security aspects need to be considered before adopting a P2P solution for the system. In respect to data, once it is released into the peer community, it is almost impossible to control it. While it is possible to protect the integrity of the data with digital digests and hash values, it may not be possible to control copying or deletion of an object. Secure communication is another problem. Most of the P2P systems use insecure protocols for communication and among numerous participating peers it may be difficult to recognize the malicious ones.

Another concern is manageability of the system. If any kind of control or tracking is required, P2P may not be the right way to go. Financial transactions are a prime example where regardless of the benefits of P2P, all involved parties would likely prefer a centralized system with strong authentication and tracking capabilities.

Finally, achieving good performance is not trivial in P2P systems, although it is a major goal. The first steps in improving performance are distributing processing load and aggregating resources, such as processing power and storage from individual peers. For distributed computing, these steps may be enough, but in file sharing and storage systems, the amount of network traffic is a significant factor. In any P2P system where searching for resources is the core function, a search engine determines the performance and scalability of the system. Initial flooding approach in Gnutella network illustrates this problem well [43], where scalability is limited by exhausting the available bandwidth with multiplying queries.

2.4 P2P Systems and Architectures

With P2P becoming a computing paradigm of its own, there is a necessity to classify it and understand the variety that exists within it. In the computer system taxonomy, P2P is placed under the category of distributed systems [31]. From the architecture perspective, we can distinguish between “pure” P2P systems, which are completely autonomous from any central control or component, and the “hybrid” ones, which include some form of centralization, even for non-critical tasks. More commonly, P2P systems are classified according to their purpose into distributed computing, file sharing, collaboration and platforms [31]. These systems are briefly described with their representative applications.

2.4.1 Distributed Computing

This type of P2P system is aimed at solving complex computing problems by breaking them into smaller tasks and executing them in parallel on a number of peers. Such computing problems include market analysis, code breaking, searching for extraterrestrial life or bio-informatics (e.g. cure for cancer). The main representative of this category is SETI@home, a project aimed at analyzing radio signals from outer space in search for extraterrestrial life [50]. This is a hybrid P2P system in which participating peers connect to the central server periodically to obtain the computing task and deliver results. The peers share processing power, but they do not interconnect.

2.4.2 File Sharing and Content Storage

File sharing P2P systems allow users to locate the files of interest according to their name and offer their own files to others. The files are classified as music, video, image, etc. The major feature of this type of system is that it allows quick location of files and a way to obtain copyrighted content at no cost.

The file sharing hype started with the centralized Napster file sharing service, which was a server farm offering a directory of music files [10]. Its purpose was to allow peers to find other peers that have the song of interest and then connect directly to one of them for download. Gnutella followed with a pure P2P architecture, creating a complex interconnected overlay network, with major clients being BearShare [4] and LimeWire [29]. The FastTrack network arrived next with Kazaa [25], the client that became synonymous with the network itself, using super peers as small hubs that offer directory services for all peers connected to it [51]. This approach facilitates much faster search and Gnutella network soon followed this model [42]. Freenet is a system where anonymous peers provide information storage to each other, with anonymity being the primary feature [31].

2.4.3 Collaboration

Collaboration is the most user-involved type of activity that can be supported by a P2P approach. Virtual workspaces allow users to stay at their workstations, but at the same time collaborate and interact in real time without moving to a conference room. This type of collaboration is especially useful for telecommuters. The P2P approach enables shared workspaces without central servers and databases or any specific infrastructure.

The simplest form of P2P collaboration is instant messaging, popularized by Jabber [17], AOL AIM, MSN Messenger, Yahoo! and ICQ messaging applications. Groove adds message board, file repository, calendar and custom modules in its collaborative application [9].

2.4.4 Platforms

To date, the number of P2P platforms remains small. The only “true” platform in existence is JXTA, whose goal is to formulate standards and provide the P2P infrastructure. Vendors of other P2P solutions, such as Groove and Jabber, provide software development kits for creating customized applications, but they are still based on the underlying product. A P2P platform must provide the infrastructure for peer connectivity, messaging, organization, etc. In some sense, .NET My Services can also be classified in this category, since it provides web service discovery using UDDI [57], service description with WSDL [59] and the development framework [35]. However, web services strongly rely on centralized UDDI servers and organization in peer groups is not supported.

2.5 Performance Evaluation of P2P Systems

The key issues in design, deployment and use of P2P systems are performance, manageability and security. This work concentrates on the performance aspect. Under the general performance umbrella, we are concerned with user, system or network perspective. The user wants fast application response and satisfaction of her/his computing needs. File sharing applications are primarily evaluated by the speed and quality of search results, which ultimately determine the user's experience. From the system perspective, the question is what resources are needed to achieve the user satisfaction. The major concern in pure P2P systems, such as early Gnutella, is the amount of bandwidth necessary for the propagation or flooding of search queries, which is of interest from the network perspective.

Most of the early P2P solutions are not built with high performance and scalability in mind [39, 43], but rather to satisfy the need to quickly provide a solution to a problem. This has prompted research into performance, scalability and characterization of P2P systems. The file sharing and storage networks provide the main field of performance evaluation of P2P systems. Several ways of measuring computer system performance are in use and they are discussed in the context of P2P systems and applications.

Analytic evaluation is used in several aspects of P2P computing, most extensively in the evaluation of object location and routing algorithms. This approach is used to show the limitations of the broadcast-based query propagation in early Gnutella networks [44]. Kant and Iyer extend the theoretical analysis of the same kind of P2P networks to draw conclusions such that it is beneficial to have distinguished nodes (super-peers) and that nodal capacity must increase rapidly to keep up with network size in order to satisfy all possible queries [22]. Analytic methods are further used to prove the efficiency of newer large-scale location and routing protocols, such as CAN [39], Chord [53], Pastry [46] and Tapestry [61], which promise object location in sub-linear time in respect to the network size. These protocols are based on distributed hash tables (DHT) and they can be used to build application-level multicast [5], distributed file storage [26] and cooperative web caching [16], as well as be a part of any P2P system's resource discovery and routing component, such as in JXTA [56]. Theoretical analysis is also used to describe the behavior of such a DHT system in the environment of continuous joins and failures of nodes [28, 53].

Simulation is the usual next step following the theoretical analysis, and it is used in almost all of the mentioned work. Simulation results more closely reflect the real-world environment and reinforce the analysis of the idealized model. Simulation is in particular suitable to demonstrate the behavior of the large-scale system, especially networks of thousands of nodes, as intended to be supported by DHT algorithms and their applications [5, 6, 16, 26, 28]. One evaluation of caching in saving bandwidth consumed by file sharing applications is also performed using simulation and reveals the great potential of caching strategy [47]. Problems still exist with simulations, since the simulators are generally custom-written for the particular application, algorithm or operating system, and the comparisons are difficult since different topologies and parameters are used. Simulations also require prior knowledge about the behavior of the network.

The next form of gathering information about P2P systems is empirical. **Traffic tracing and probing** is used to collect the most data and best describe the popular file sharing networks, Gnutella and Kazaa. Traffic tracing consists of recording the network packets at one or more locations, and probing is conducted by connecting a special peer into the network to collect data about other peers. A large traffic characterization study was conducted based on a campus-level trace of several content delivery networks and among them the P2P file sharing traffic was recorded [47]. This study showed the enormous amount of traffic that file-sharing produces, daily patterns of usage, and that P2P traffic poses a threat to the campus network bandwidth. Using crawlers for Napster and Gnutella networks, another study concluded that a small number of peers serve most of the content and act like servers and that a significant portion of the participants are free-riders, which do not offer any content, rather just download files [48]. A similarly conducted study shows that the Gnutella overlay network does not match the underlying physical topology and, therefore, inefficiently uses the Internet infrastructure [43]. Additional characteristics of Gnutella are shown to be bursty traffic patterns, query locality and a potential to use caching to improve performance [30].

A related empirical method of performance evaluation is **benchmarking**, which provides the most accurate and realistic results. Benchmarking of the prototype implementations on a wide-area test-bed is employed as a final step to confirm the simulation results for Pastry [46] and Chord [53]. This method is also used to test the hypothesis about benefits of caching in file sharing networks in [27] and to evaluate JXTA-wire, a component for many-to-many communication [3]. A bootstrapping function in Gnutella is evaluated using benchmarking and crawling, which revealed the impact of application-dependent implementation on the performance of this operation [24]. Still, more features in Gnutella or other file sharing networks could be benchmarked. For example, a super-peer's performance in locating content, latency in connecting to other peers, resource usage of regular and super-peers are all metrics of interest for which results are not available.

2.6 Object Location and Routing in P2P Systems

Object location became a central performance issue of P2P research due to the prevalence of file sharing among the public P2P systems and the scalability issues of their early versions. Quick object location is achieved by one of two ways, replication or efficient routing. In flooding-based file sharing systems, replication brings the files effectively closer to the interested peers. An additional benefit of replication is the possibility to download parts of a file from multiple peers in parallel and thereby reduce download time [7, 25]. The replication level depends only on the popularity of the file. In Freenet, files are cached at every peer on the path to the destination, achieving the same goal.

DHT protocols take a different approach by using an efficient algorithm to find the location of the file. The DHT search mechanism requires the user to know the exact identification of the object before submitting a query, which is extremely strict compared to the flooding or directory-based approaches that require only a part of the object name (wildcard search). The main concern is that DHT systems require a stable cooperative peer community, which is not realistic, so they still need to prove their feasibility in the real world. The choice of the approach to employ in a particular P2P system depends on the requirements of the application. For example, JXTA uses a DHT-based approach to connect only a part of the system, its rendezvous network, as explained in Chapter 3.

2.7 Summary

This chapter gave the background on P2P systems. Although P2P is becoming a rich family of distributed systems, there are only a few applications widely used and accepted, and the most popular and best understood belong to the file-sharing category. Moreover, the main purpose of the applications such as Kazaa, LimeWire and eDonkey is not more efficient use of resources, self-organization or any other major benefit of P2P paradigm, it is rather an easy and free access to the copyrighted music or video, and a way to avoid legal liability.

The available performance results show a large heterogeneity of peers in terms of network bandwidth, lifetime, amount of shared data and willingness to cooperate. Such findings demand that future P2P applications and protocols, including JXTA, be designed to provide good performance, high scalability, and adaptation to heterogeneous environments.

Chapter 3

JXTA Overview

Most of the P2P applications have a very specific way of dealing with peer and resource discovery, communication and resource sharing. The common P2P systems focus on a single purpose, with one or few different applications supporting it. Major file-sharing, collaboration and computation systems cannot interoperate. Even within the file-sharing family, distinctive and mutually exclusive networks have formed, best known as Kazaa, Gnutella and eDonkey networks.

Project JXTA [36] has joined the P2P movement with a novel approach. Instead of providing a solution for a specific P2P application domain, JXTA offers generic building blocks for the development of any type of P2P system, from collaboration [33] to parallel computation [58]. JXTA defines protocols for core P2P operations, such as discovery, messaging and group organization, which are necessary for all P2P applications.

Over three years of development in an open-source community have produced a solid API, largely tested in academic environments as a platform for prototyping various P2P concepts. The statistics from June 2004 indicate that the JXTA community has grown to over 17,000 members [36]. The existing applications of JXTA range from RDF-based resource discovery and retrieval [34], to agent-based P2P systems [2]. The popularity of JXTA has spread into the market environment and some commercial products are already available [37].

As a set of protocols, JXTA is independent of the programming language, operating system, device or underlying network transport [55]. JXTA uses open standards as its interoperability foundation, which is best represented by its dependence

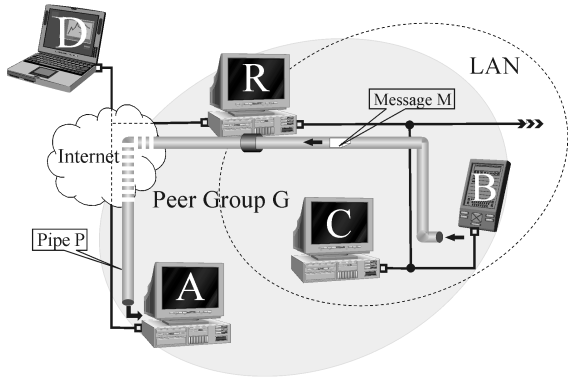


Figure 3.1: JXTA peer network ([13])

on XML. The reference implementation is available in Java. C, Python, Perl and other language implementations are provided as community projects [36].

In general, P2P applications must handle intermittent connectivity, dynamic IP addresses and unstable network topology. Therefore, the developers face a challenge of designing applications that are independent of DNS and IP addressing and identification. This is where JXTA introduces a virtual network layer, on top of the existing transport protocols, with its own addressing and routing, facilitating interaction in this dynamic environment [55]. This virtual network is able to cross barriers like firewalls and Network Address Translation (NAT), and establish peer communities spanning any part of the physical network.

The main concepts in JXTA are described in the following sections, with reference to the example in Figure 3.1, where a network of five peers is shown. The following scenario is assumed: peer A provides a weather forecast service, and peer B needs to discover it and request the current forecast report.

3.1 Peers and Peer Groups

The JXTA virtual network consists of several kinds of peers [38]. Most of them are *simple* or *edge* peers, usually desktop computers connected by a LAN or modem to the Internet, such as peers A, C and D. Small devices, such as peer B, are *minimal* peers, since their resource constraints would most likely disallow full functionality. They use help from *proxy* peers, such as peer C, for caching, search and discovery. *Rendezvous*

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
<Id>
        urn:jxta:uuid-
59616261646162614E50472050325033BC987F4F5B744EF2A88EACCA479A1E1F04
 </Id>
 <Type>
        JxtaUnicastSecure
 </Type>
 <Name>
        peer1JxtaUnicastSecureSenderPipe
 </Name>
</jxta:PipeAdvertisement>
```

Figure 3.2: JXTA secure pipe advertisement

peers are usually more powerful peers, with well-known DNS name or stable IP address. They act as the caches of information about the peer group resources and help in resolving the connections between edge peers. *Relay* peers learn and provide routing information and pass messages between peers separated by a firewall or NAT. In Figure 3.1, peer R acts as both relay and rendezvous.

Peers organize into peer groups and all communication is constrained to the group members. Peer groups are not limited to the physical network boundaries. In the network from Figure 3.1 peer D does not receive messages from group G, because it is not a member, although it may use the same rendezvous or relay. Advertisements, pipes, messages, rendezvous and relays are crucial entities that make the JXTA network function and as such are relevant for the performance study.

3.2 Advertisements

All entities in JXTA, including peers, groups, pipes and services, are represented by advertisements, which are well-defined XML documents [21]. Advertisements carry a unique random ID number of the resource or entity they represent, and optional additional information, such as human-readable name and description (Figure 3.2).

Peers use advertisements to learn about other peers and services they provide. Advertisements have a lifetime, after which they are considered stale and purged. A publishing peer is responsible for “refreshing” or republishing its expired advertisements. The lifetime mechanism is important for automatic repair of the network, in case of peer departures and failures. A major peer operation is to purge its local cache of stale advertisements upon start-up. This prevents a peer from attempting to access non-existent peers and services.

Peer A from Figure 3.1 publishes its weather forecast service advertisement, which other peers cache for a specified lifetime. During this lifetime, potential consumer peers, such as peer B, can find the service and access it. Before the lifetime has expired, peer A should republish the service advertisement. If peer B joined the network after the advertisement was published, it can search and discover the advertisement from the network.

Publishing, discovery and exchange of advertisements is an essential step in the process of connecting a JXTA peer network. Efficiency of advertisement processing and management impacts the performance of operations on the resources represented by advertisements.

3.3 Pipes and Messages

JXTA pipes are a fundamental abstraction used for inter-peer communication. JXTA peers pass messages through pipes, virtual channels that consist of input and output ends. Peers bind to one end of the pipe, and when both ends are bound, messages can be passed. Pipes are not tied to the physical location, IP address or a port. Instead, a pipe has a unique ID, so peers can carry their own pipes with themselves even when their physical network locations change. At runtime, a pipe end is resolved to an endpoint address to which it is currently bound.

In Figure 3.1, peer A must open a pipe input end to receive forecast service requests, and publish the pipe’s advertisement, either separately or embed it with the forecast service advertisement. This informs other peers where to connect if they want to send a request. At runtime, peer B obtains the pipe advertisements and queries the network for the peer who has opened the pipe’s input end. Once peer A has responded, peer B can open the pipe output end and send the request.

Pipes are unidirectional and unreliable by definition, but bi-directional and reliable services are provided on top of them. Two operation modes of pipes are defined, unicast and propagate. Unicast pipes serve for one-to-one communication, connecting two peers. Propagate pipes connect one sender peer to many receiving peers. Pipes are asynchronous, and message elements such as unique IDs are used for sequencing.

Unicast pipes can be combined to achieve many-to-one communication. From the receiver's perspective, a single operation is required to open the input end of the pipe. Multiple senders can discover the same pipe and open their output ends for communication. Senders see the pipe as one-to-one connection, and handling of multiple connections at the receiver's side is completely transparent to the programmer. A secure pipe operates in unicast mode, with the additional security provided by TLS layer [55].

A propagate pipe is used for one-to-many communication; leveraging either IP multicast on the subnet or the rendezvous peers for message propagation. A sender commonly opens the output end of the pipe first and starts sending, whereas receivers connect to the propagate pipe by opening their input ends to receive any messages transmitted by the sender.

The described three kinds of pipes are defined in the core JXTA protocols and they will be referred to as core pipes. Two additional and useful components also exist in JXTA: bi-directional pipe and JXTA socket. Both are included in the performance evaluation as non-core JXTA services. These services are the additions to the JXTA platform that offer additional features on top of the core specification.

The purpose of the bi-directional pipe is to provide bi-directional communication within a single messaging object, since core pipes are unidirectional. A bi-directional pipe features an API that is similar to a well-known Java socket API. Under the abstraction layer, the original version of the bi-directional pipe used two core unidirectional pipes. The newer version (2.2) used in this study is implemented differently. The connection is opened using a unicast pipe for one direction, and then the reverse direction link is established using an internal endpoint service, which is normally not directly used at the application layer. The normal pipe resolution is bypassed for the reverse direction. All connection code is still hidden under the single method invocation.

Similar to the core pipes, the use of bi-directional pipes assumes that one peer initially creates an input endpoint (server), to which another peer (client) will connect. The difference in the API requires a programmer to implement own multi-threaded server, unlike for core pipes that have a built-in handling of multiple connections.

According to the available documentation, the JXTA socket is an optimized bi-directional pipe. It uses a standard JXTA pipe service for the initial connection and endpoint addressing for the reverse connection, similarly to bi-directional pipe. The JXTA socket provides reliability directly and has no limit on the transferred data size (it transmits in chunks of 16 KB by default). The main difference between a JXTA socket and other pipes is that it provides the same API as the standard Java socket classes, and hides the underlying pipe implementation. This includes the methods for obtaining the input and output streams as the primary communication API. The JXTA socket directly allows byte-level access to the transmitted data, which offers highest flexibility to the programmer in designing a communication protocol. Therefore, JXTA messages are not the intended primary unit of data transmission over JXTA sockets at the application layer.

JXTA messages are XML-documents composed of ordered elements [21]. The elements are name-value pairs, and they can carry any type of payload (Figure 3.3). JXTA uses source-based routing and each message carries its routing information as a sequence of peers to traverse. The peers along the path update this information. The routing elements tend to get large, primarily due to the 256-bit peer ID [21], composed of own and the group ID. This implies that even an “empty” message, with no application-specific payload, can easily reach 1 KB in size, affecting the performance of the message exchange.

3.4 Rendezvous and Relay Peers

To support resource binding and exchange of messages across networks and firewalls, two special concepts exist in JXTA, rendezvous and relay peers. Rendezvous

```
private void doGetPostings(String forum, OutputPipe op)
{
	Vector postings = ServerManager.getPostings(forum);
	try {
		Message msg = pipeSvc.createMessage();
		msg.setString(Global.CONNECT_TAG, "Connect");
		msg.setString(Global.REQUEST_TAG, Global.POSTING_LIST);
		msg.setString(Global.FORUM_TAG, forum);
		// Use BAOS as an underlying stream for serialized object
		ByteArrayOutputStream baos = new ByteArrayOutputStream(512);
		ObjectOutputStream oos = new ObjectOutputStream(baos);
		oos.writeObject(postings);
		oos.flush();
		MessageElement el = msg.newMessageElement(
			Global.POSTING_TAG,
			new MimeMediaType("application/octet-stream"),
			baos.toByteArray()
		);
		msg.addElement(el);
		op.send(msg);
	} catch (java.io.IOException e) {
		System.out.println("Failed to send message");
		return;
	}
	op.close();
}
```

Figure 3.3: Passing serialized Java objects as JXTA message elements

peers cache information for their peer group, facilitate search and discovery, propagate messages and scope the advertisement query recipients. They also provide resolving operations, such as peer or pipe name resolution to an IP address [55].

Peer R acts as a rendezvous peer for group G in Figure 3.1. When peer A publishes the service or pipe advertisement, it is indexed at peer R so other peers can find it. When the peer B searches for the forecast service, peer R can locate it at peer A or propagate the query further.

The concept of a rendezvous peer in JXTA is similar to the concept of super-nodes in file-sharing P2P networks, such as Kazaa and Gnutella. The super-nodes act as file indices and facilitate quick and efficient search. Although the search for advertisements in JXTA is not the same as the search for files, the role of the rendezvous is no less important.

Relay peers, on the other hand, pass messages and store routing information. Although JXTA messages contain routing information, relays are used when communication has to go through a firewall. Relay peers can also spool messages for unreachable peers and serve as bridges between physical networks [55]. For example, since peers A and B are on different networks, they need relay R to pass messages between them, if they cannot connect directly.

Any JXTA peer can become a relay or a rendezvous, but this usually depends on hardware and bandwidth constraints and security policies. Enterprise installations behind a firewall or NAT usually expose one public rendezvous/relay for connections from outside peers.

3.5 JXTA Rendezvous Network

Rendezvous peers serve several purposes and can potentially be subject to high message loads. The response time, message and query throughput and advertisement cache management are all important performance factors for a rendezvous peer. Discovery queries and responses are the special kinds of messages exchanged generally between an edge and a rendezvous peer or two rendezvous peers. It is also possible that two edge peers exchange the discovery messages, if a rendezvous is not available and both edge peers are on the subnet. Still, according to JXTA 2.x specifications and implementations [21], an edge peer is made a rendezvous peer if the dedicated rendezvous is not available. Unlike the initial version, JXTA 2.x mandates the use of a rendezvous, moving towards the hierarchical network structure similar to Kazaa, Gnutella 0.6 and more recent DHT systems [40].

Rendezvous peers in JXTA create a sort of a sub-network within the whole peer population that includes edge, minimal and relay peers. This Rendezvous network has special properties in JXTA 2.x [56]. For example, rendezvous peers do not replicate edge peers' advertisements, nor propagate queries to the edge peers, like in JXTA 1.0. Rendezvous peers now use a DHT to maintain the index of advertisements across all known rendezvous peers within a peer group. This index is called *Shared Resource Distributed Index* (SRDI) and it represents the new search and discovery model in JXTA.

The search for a resource using SRDI is similar to Chord [53]. The query is propagated between rendezvous peers until the correct index is found, at which point the edge peer who stores the resource is asked to contact the sender with a positive response. The propagation is based on the Rendezvous Peer View (RPV), which represents each rendezvous peer's view of other known rendezvous in the peer group.

The main difference between SRDI and other DHT protocols and systems is that the RPV consistency across rendezvous peers is not enforced in JXTA. Rendezvous peers occasionally exchange their RPV, which becomes more or less consistent depending on the join and leave (churn) rate of rendezvous peers. The Rendezvous network is therefore "loosely-consistent". It is inevitable that this design causes more search misses due to inconsistency, but the remedy for this is found in the *limited-range walker* that linearly searches adjacent rendezvous peers to the original DHT target. The walker behavior is based on the rule that each individual index (hash value) is replicated at the rendezvous peers adjacent to the original location at insertion time.

Such hybrid design is based on the expected high churn rate and seen as a good compromise between consistency and avoiding high index maintenance cost. Edge peers normally maintain a connection to one rendezvous, although they may be aware of more. A leaving rendezvous would trigger a dynamic recovery mechanism at edge peers to use another known rendezvous.

3.6 Search Models in JXTA

JXTA supports three models for resource search and discovery:

- Centralized.
- Distributed with query flooding.
- Distributed with SRDI.

JXTA 1.0 allows for centralized and distributed search with flooding, whereas JXTA 2.x allows for a centralized and distributed model with SRDI to be used. The centralized model in JXTA refers to the peer group where all resource advertisements are located on the rendezvous peer. In a distributed model, advertisements are located on individual edge peers, and rendezvous peers only facilitate the search and discovery. A rendezvous peer in JXTA 1.0 floods the received query to the connected edge and rendezvous peers, looking for a response. In JXTA 2.0, a rendezvous peer maintains the SRDI, so the query is either immediately sent to the peer who stores the resource, or forwarded to another rendezvous, according to the index map.

3.7 JXTA Protocols

The Project JXTA specifies six protocols divided into two categories: Core Specification Protocols and Standard Services Protocols.

3.7.1 Core Specification Protocols

Since JXTA protocols are designed for any networked device, there are only two required protocols to be implemented on every peer, as defined by the JXTA Core Specification protocols:

The *Endpoint Routing Protocol* (ERP) is the protocol that allows a peer to discover a route for sending a message to another peer. In the absence of the direct route between two peers, a peer can find an intermediary to route the message to the destination peer.

The *Peer Resolver Protocol* (PRP) is used for sending a generic resolver query to one or more peers, and receiving a response (or responses) to the query. The PRP protocol distributes the generic queries to one or more handlers within the group and matches them with the corresponding responses.

3.7.2 Standard Services Protocols

The JXTA Standard Services Protocols are optional protocols and behaviors that are strongly recommended in order to create a complete JXTA implementation. Implementing the recommended services provides greater interoperability with other implementations and broader functionality.

The Standard Services Protocols specification defines four protocols:

The *Rendezvous Protocol* (RVP) is the protocol by which peers can subscribe to a propagation service or provide one. Peers can be rendezvous peers, or standard peers that are listening to rendezvous peers. RVP therefore allows rendezvous functionality and it is used by the PRP to propagate messages.

The *Peer Discovery Protocol* (PDP) is responsible for publishing and discovering advertisements. PDP uses the PRP for sending and propagating discovery requests.

The *Peer Information Protocol* (PIP) is the protocol by which a peer may obtain status information about other peers, such as state, uptime, traffic load and capabilities (PIP also uses the PRP).

The *Pipe Binding Protocol* (PBP) is employed to establish a virtual communication channel or pipe between one or more peers. Using PBP a peer binds pipe ends to a physical endpoint address. PBP uses the PRP for sending and propagating pipe binding requests.

More details about the JXTA protocols can be found in the JXTA protocol specification [21].

3.8 Existing Work on JXTA Performance

This section presents the existing work and results on JXTA performance from the available literature and online resources. The Project JXTA matured in its design and implementation after three years of development. However, very little is available in terms of performance results and characterization of JXTA.

Some performance measurements are available for components of early releases of JXTA, and mostly in the context of a particular application. A higher-level JXTA service, JXTA-wire (many-to-many pipe) was evaluated for support of the type-based publish-subscribe approach for building P2P applications [3]. The JXTA propagate pipe was compared to the alternative solution for high-speed communication within peer groups [18]. Additional results found JXTA to have poor messaging performance compared to TIBCO Rendezvous [54].

My work preceding this thesis includes the investigation of the peer discovery and unicast pipe performance in the context of a P2P discussion forum system [11].

These evaluations used benchmarking and all results reflected the performance of the JXTA 1.0 implementation. The results [11] indicated that rendezvous peers improved the discovery performance within a peer group and that group size and query rate affect the response time, as well as that message passing reliability depends on both the sending rate and network distance. Although the evaluation was conducted in the context of a specific P2P forum application, it showed that benchmarking is an excellent method of learning about JXTA performance characteristics.

The major source of performance data is the JXTA community Bench project web site [20]. It offers a small set of results on pipe and discovery performance, and only recently, it started to include several peer and rendezvous configurations. The results are presented using a summary table and graphs, with no discussion. The main purpose of the Bench project is to track progress through major and minor JXTA releases, and guide developers in optimizing the implementation. The comparisons are made between the latest and previous versions, without any comments on the evolution of performance over longer time frames, for example from one version to another. The Bench project presents results from a single controlled environment, which is an isolated LAN connecting several high-performance computers.

In terms of the actual presentation of the Bench project results, the time-series graphs are overcrowded with several overlapping measurements, which make it difficult to distinguish patterns of behavior. Even when some interesting patterns are observed, such as spikes or flat lines, there is no discussion or explanation. It is possible to refer to the Benchmark plan for reference, but the suggested and actual configurations do not always match, especially with the latest results.

The only other targeted performance evaluation was provided in [49] for early versions of JXTA that showed the performance of small-size messaging over pipes for different operating systems and transport protocols. This evaluation concluded that the early JXTA releases were characterized by low reliability and high operational latencies.

All of the available results were obtained by benchmarking, which is still the only possible method for the JXTA platform. The JXTA protocol specification [21] does not clearly specify any algorithms suitable for analysis or simulation, and there is no widely deployed public JXTA network suitable for probing and traffic tracing. The performance of JXTA is dependent on the details of its implementation, and the Java-based reference implementation is the only one that is complete and updated with the latest design decisions. The Project JXTA features no public design documentation suitable for performance evaluation either. Therefore, benchmarking of its reference implementation presently seems the only way to go.

This research aims to include the analysis of the following factors that lack in the results available from the Bench project and other sources:

- ❖ Effect of message size and composition on pipe messaging performance
- ❖ Effect of query load on rendezvous response time and scalability
- ❖ Latencies of typical peer operations under different transport protocols and rendezvous configurations
- ❖ Comparison of pipe performance through major JXTA versions

The apparent incompleteness of the performance evaluation provided by the Bench project was one of the main factors that motivated this thesis research project. The intent is not to criticize the Bench project, rather to complement it by looking in-depth at JXTA components and features, and systematically conduct the performance analysis.

3.9 Summary

JXTA is an evolving P2P platform, developing a rich infrastructure for building interoperable P2P applications. It includes protocols, messaging, identification and security features. Although it promises powerful and reliable solutions, it provides very limited information on the performance, scalability, and applicability to different types of applications and devices. A performance model is missing since different users and researchers are evaluating only what they need in their application-specific context.

The complexity of the architecture of JXTA calls for a thorough performance evaluation and analysis, based on a model that would cover all relevant components and environments, which is exactly the objective of this thesis.

Chapter 4
Research Goals

This chapter summarizes the research goals and presents the outline of the thesis work in three phases. The goals are presented as an overview with details discussed in the following chapters.

This research has the following goals:

- Design and develop a JXTA Performance Model and Benchmark Suite.
- Obtain performance bounds and identify the bottlenecks of the components and systems built on the JXTA platform, based on the benchmarking results in a variety of network configurations.
- Formulate the deployment patterns that achieve the desired performance levels of the JXTA system.

The accomplishment of the goals proceeded in three phases:

- *Phase 1* consisted of establishing the initial JXTA Performance Model and development of the basic benchmarks. This phase also included the investigation of the core JXTA components in a LAN environment and one-to-one communication.
- *Phase 2* included refining of the Performance Model, completing the Benchmark Suite, evaluation of non-core JXTA services and extending the performance testing to the wide-area network environment and multiple sender and receiver scenarios.
- *Phase 3* involved the analysis of the performance results, formulation of lessons learned from this research and discussion of goal accomplishments.

Chapter 5 covers the details of the development and refinements to the Performance Model and Benchmark Suite and Chapter 6 provides the analysis of the performance results. The characteristics of the JXTA platform drawn from this study are discussed in Chapter 7.

Chapter 5

Performance Model and Benchmark Suite

This chapter describes the development of the JXTA Performance Model and the Benchmark Suite. The initial model and the basic benchmarks were used to collect the performance results of core JXTA components constrained to the LAN environment and one-to-one communication. The model is later refined and the Benchmark Suite completed to accommodate the non-core JXTA services and multiple sender and receiver scenarios.

5.1 Initial JXTA Performance Model

The architecture of JXTA is complex, consisting of six standard protocols, three basic ways of message passing and a network structure with several types of peers. From the performance perspective, all of these aspects are important and need to be considered in the performance analysis. It is not easy to evaluate the performance of a complex system; therefore, a model is required to systematically cover all the relevant components. Since P2P systems are still in their infancy, performance metrics and criteria are not yet standardized.

The initial JXTA Performance Model is consistent with the Benchmark Plan developed by the JXTA Bench project community [20]. The Performance Model defines much wider framework with more performance metrics, including the typical peer operations, pipe message and data throughput and relay message throughput. Investigation of numerous combinations of parameters is also recommended: various rendezvous and relay configurations, search and discovery models, message sizes, compositions, etc. The results collected using the Performance Model are useful for more than just the platform development efforts. In particular, P2P system designers and users benefit from the identified optimal peer and network configurations for their specific purposes. The programmers can derive guidelines for efficiently implementing communication and discovery functionality, and the researchers are provided with the realistic input parameters for the simulations of JXTA-based networks.

The initial JXTA Performance Model consists of the following components and performance metrics:

- Latency of typical peer operations
- Pipe message round-trip time
- Pipe message and data throughput
- Rendezvous query response time, throughput and reliability
- Relay message throughput

This model was originally proposed in [12], where the metrics were discussed and evaluated. The following sections describe the components of the model and give arguments for their inclusion.

5.1.1 Latency of Typical Peer Operations

Figure 5.1 shows a series of operations that a peer needs to perform to join and participate in a JXTA network. Depending on the application and type of peer (edge, rendezvous or relay), the number and order of these operations may vary. The operations are mandated by the design of the JXTA protocols and their implementation. Even the simplest peer operations are implemented with a high level of abstraction. It is expected to find a high performance penalty associated with layers of abstraction, especially in terms of latency and response delay. In addition, a peer may repeatedly and frequently perform some operations, which then aggregate to a large performance cost.

In contrast to the P2P file-sharing peers [8, 25], JXTA peers are significantly more complex. The versatility of JXTA and its applicability to any type of P2P application comes at the price of the increased complexity. A typical file-sharing peer connects to one or more other peers of the same kind, exchanges the shared file list and queries the network for content. The initial peer discovery and file transfer are typically

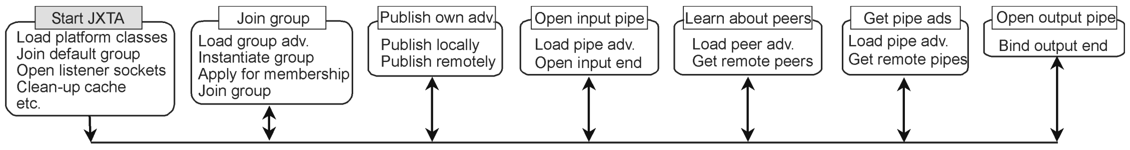


Figure 5.1: Composition of the typical peer operations ([12])

out-of-band operations, unlike in JXTA, which uses multicast discovery on a subnet and pipes for message and content exchange. The additional features introduce more basic and costly peer operations.

Most of the high-level JXTA peer operations are actually sets of distinct basic steps, as shown in Figure 5.1. From the performance perspective, the developers and users should be interested in the cost of these operations due to the following reasons:

- ❖ High cost of the basic steps involved to complete an operation,
- ❖ Frequency of steps or operations performed during the peer's lifetime in the network.

To simplify the evaluation, only the costs of the typical high-level operations are measured in this study. The decision on what is considered typical is based on the review of two JXTA applications: a P2P forum system [11] and MyJXTA [33].

Figure 5.1 shows the typical high-level operations in the relative order a peer performs them upon startup:

1. *Start the JXTA platform*, to initialize the environment for running JXTA protocols and services.
2. *Join a peer group*, according to user preferences or common peer services, and to enjoy a more secure and efficient environment.
3. *Publish own advertisements*, to make peers aware of the presence and available resources.
4. *Open an input pipe*, to receive messages from peers.
5. *Learn about other peers*, who participate in the same group and share common resources.

6. *Obtain pipe advertisements*, to discover available communication channels of other peers.

7. *Open output pipe* to send messages to other peer(s).

Starting the JXTA platform loads a class library, which depending on the configuration of the peer, may involve access to the local disk, network file system (NFS) or the Internet. This operation builds the data structures to support the JXTA platform, cleans up local cache and opens listener sockets. The basic platform startup cost is explored in [13] with respect to JXTA and Java Virtual Machine (JVM) versions. This study expands the existing work in regard to cache size and rendezvous connection.

During the platform startup, all peers instantiate and join NetPeerGroup by default, which is the universal worldwide group. Peers may choose to stay in this group, or join other sub-groups. Depending on its configuration, an edge peer may also connect to the rendezvous and relay peers during startup. For any peer, the platform startup is the first, complex and essential operation to perform and therefore considered relevant for the performance study.

Joining a group is usually the next step after startup, performed by loading a peer group advertisement from the cache, instantiating a peer group object, and then applying for membership using a credential. Obtaining a membership may involve different delay costs, according to the security policies of the peer group. If a peer is joining the group for the first time, it must discover the peer group advertisement from the network or initiate a group by creating a new advertisement. Rendezvous peers are normally the ones that initiate a peer group, whereas edge peers discover it once and then they just reload it from cache or a file. A peer is not limited to a single group and therefore performs this operation for each group it wants to join.

A peer *publishes its own advertisement(s)*, thereby announcing its availability to provide a service to other peers. JXTA allows for two types of publishing, local and remote. Local publishing puts the advertisement in the local cache, from which it is sent out when discovery queries arrive. In this case, it is up to the other peers to send a query to obtain the advertisement. Remote publishing sends out the advertisement index towards other peers through a rendezvous peer. Depending on the frequency of publishing and the number of connected peers, this may turn out to be costly in terms of network traffic and processing on the peers and rendezvous.

Opening an input pipe refers to the creation of a class instance that represents an input end of a unidirectional pipe. A pipe end is created using a pipe advertisement, which is typically first read from a hard disk. These two steps are treated as a single operation because they are usually performed together. A peer creates a new pipe advertisement only if the old one was destroyed or never existed. A good JXTA citizen would always reuse own advertisements, which other peers may have already discovered and cached.

The *learning about the peer group* operation depends on the actual application. For a shared computing system, it may be important to know who is active in the group to properly distribute load. On the other hand, for a distributed discussion forum application [11], a peer may be interested in just any other peer that is hosting a given forum. Although this operation may not satisfy a strict definition of “typical”, it is included for both completeness and its high cost in case of remote discovery, especially when a peer group is created ad-hoc, without a rendezvous. The peer can search in its local cache for known peers (advertisements) or query the network for remote peers.

Obtaining a pipe advertisement is probably the second most frequent operation executed by a peer; the most frequent would be sending a message. A pipe advertisement can be discovered from a network, retrieved from the local cache, user-specified file, URL, etc. Usually, a pipe advertisement is embedded within another advertisement published by a peer, such as a service advertisement. It is expected that a peer would use the local cache to avoid the cost of discovery from the network and the results section shows the significant magnitude of this saving.

Nevertheless, it is important to obtain measurements of discovery cost because caching is not always an option. For example, JXTA advertisements have an associated lifetime, so unless a publisher refreshes the advertisement, it would be purged from other peers’ caches, forcing them to use remote discovery. Caching is not required by the JXTA protocols, so some peers may elect not to implement it, due to various reasons such as resource constraints or security. Another reason for measuring discovery latency for some deployment scenario is to determine the appropriate frequency of sending discovery queries. It would not be desirable to have peers send queries faster than responses could arrive. This would generate unnecessary network traffic in duplicate queries and redundant responses, as well as wasted CPU cycles of requesters, responders and intermediate peers on the path.

Opening an output pipe means binding to the output end of a pipe for which another peer has opened the input end. No option is available for this operation, it is required before sending a message and it may be costly. Binding a pipe end involves the creation of network connections between peers, several in case of relays, implying latency.

The operations discussed are typical of group-structured or hierarchical P2P networks and systems and also apply to the well-known file-sharing systems [8, 25]. Since JXTA is designed and implemented with group-based structure in mind, it is desirable that peers follow this idea.

5.1.2 Pipe Message Round-Trip Time

A JXTA message is a unit of data transfer over the JXTA pipes. Message round-trip time (RTT) is, therefore, the basic metric used to evaluate JXTA pipes and the communication sub-system. The two most common kinds of messages are the application-specific messages exchanged over pipes by the edge peers and discovery query-response messages exchanged between edge peers and rendezvous or two rendezvous peers. The messages exchanged between edge peers are referred to simply as messages. The discovery query-response messages are referred to as queries or responses, as appropriate to the context.

JXTA messages are generic and may carry any type of application-dependent payload. While Gnutella has a small and well-defined set of messages, JXTA provides a generic XML-based template for message exchange, which is capable of producing a wide variety of message sizes and compositions, ultimately affecting the JXTA application performance. Messages are transferred through pipes, which allow unicast (one-to-one), secure (one-to-one encrypted) and propagate (one-to-many) communication.

5.1.3 Pipe Message and Data Throughput

Message throughput reflects the maximum number of messages that can be transmitted one-way through a pipe without loss. In case of JXTA pipes, throughput is affected by both the efficiency of the protocol implementation and the reliability provisions. The pipe reliability is largely determined by its queuing policy. Messages are queued before sending at the sender's side, as well as on the receiver's side just before delivery to the application. The queue can amortize bursty traffic, but if it overflows, the oldest message is dropped. As different layers of abstraction and overhead are introduced, the throughput may suffer. It is expected that message queues impact throughput more than the processing overhead.

The data throughput shows the transfer limits in bytes, rather than messages. Since JXTA messages can be of various sizes, it is expected that the larger messages achieve higher data throughput, assuming a relatively stable control overhead per message. However, this depends on the message composition in terms of number of (XML) elements and the element payload size.

JXTA networks are intended for all digital devices, including sensors. It is foreseeable that some systems implemented in JXTA would depend on reliable exchange of thousands of messages per second. Therefore, the message and data throughput are important to measure. This discussion applies to both application-specific messages and discovery queries.

5.1.4 Rendezvous Query Response Time, Throughput and Reliability

Rendezvous peers connect the peer network by facilitating discovery and resolving resource names to physical network addresses. Their performance reflects on the whole peer group, especially the query handling efficiency. Query response time and throughput are the main performance metrics of interest. The query response time measures the time between issuing the request and receiving the response at the sender. The query throughput measures the ability to process a quantity of requests per unit of time and indirectly determines the size and traffic load of a peer group a rendezvous can support.

The reliability of a rendezvous peer is represented by the query loss rate and variation in response times. The query loss is expressed as a percentage of dropped messages from the total amount of issued requests. High loss rates indicate a requirement for an additional rendezvous peer, resizing of a group or more controlled querying. Variations in response times show the likelihood of receiving a response within some time frame. High variation means that a rendezvous has some difficulty in processing queries. A user will also have a difficulty deciding whether to wait or retry searching. Stable throughput, low loss rate and low variation reflect a good functioning and predictable P2P group or system.

5.1.5 Relay Message Throughput

The main purpose of a relay peer is to cache routes and pass messages between other peers that cannot establish direct connections. As the use of firewalls and NAT increases in the Internet, the connectivity becomes an issue, which makes relays more important. Relays are potentially exposed to large amounts of traffic and their scalability and performance affects the entire peer group.

Relays may act as intermediaries between any combination of edge, rendezvous and even other relay peers. Therefore, they affect the RTT and throughput of any type of JXTA messages and pipes.

5.2 Refining the Initial Performance Model

The application of the initial JXTA Performance Model revealed the need to refine the model prior to proceeding with the benchmarking. The initial model guided the benchmarking of the core JXTA components in one-to-one communication on a LAN [12, 13]. The obtained results are included in the analysis throughout the following chapter.

The purpose of refinement is to better group the JXTA components and more precisely specify all parts of the model. The improvements and evolution of JXTA, especially starting with version 2.0, require the changes. They include many-to-one and one-to-many communication, multiple rendezvous peers per group, recovery after rendezvous failure, rendezvous resource usage, non-core services and WAN deployment. The two additional non-core services are chosen to evaluate because they are new and promoted as more efficient and easier to use than core pipes. In particular, the bi-directional pipe efficiently exploits the bi-directional character of TCP within a single pipe object. The JXTA socket offers a standard Java socket API built on top of the JXTA pipes and includes messaging reliability. The refined model is formulated as follows:

- Latency of typical peer operations
    - Ad-hoc configuration
    - Rendezvous configuration
    - Parameters: peer operation, rendezvous location and protocol (TCP, HTTP), advertisement cache size
- Messaging performance
    - Core pipe message RTT and throughput
        - Unicast pipe: one-to-one and many-to-one
        - Secure pipe: one-to-one and many-to-one
        - Propagate pipe: one-to-one and one-to-many
    - Non-core service message RTT and throughput
        - Bi-directional pipe: one-to-one and many-to-one
        - JXTA socket: one-to-one and many-to-one
    - Parameters: message size, message composition, number of senders or receivers
- Rendezvous performance
    - Response time, reliability (loss, variation, recovery) and resource usage
        - Discovery in a single-rendezvous group
        - Discovery in a multiple-rendezvous group

        - Parameters: search model, group size, query rate, positive/negative queries, response threshold, peer topology and query space

- Relay performance
    - Effect on message RTT and throughput
    - Parameters: relay protocol (TCP, HTTP) and number of relays

In this study, all components of the model are evaluated in both the LAN and WAN environment (omitted from the model outline for clarity). The refined model better reflects the configurations that need to be considered for benchmarking. Specific applications may require different scenarios within the same model component, so the model is kept general enough to accommodate special cases. For example, the discovery in a multiple-rendezvous group could mean two rendezvous peers, each serving tens or hundreds of peers for a corporate scenario involving two separate networks. On the other hand, the network may contain tens of rendezvous, each serving a few peers in case of an application for small satellite offices.

5.3 Benchmark Suite

This section gives an overview of the Benchmark Suite by first describing the basic benchmarks and continuing with the additions for non-core services and multiple sender/receiver scenarios. The high-level design and some technical issues are discussed, as well as the metrics and results that the benchmarks can provide. All benchmarks consist of the sender and receiver peer. The receiver peer normally starts up and opens the various pipes, to which a sender connects. The message transfer is then initiated for the set number of samples. The propagate pipe test peers start in reverse order, according to the nature of the pipe.

Rendezvous performance benchmarking uses the modified code from the JXTA Bench project. There was no need to develop the identical benchmark, especially since the existing one has the ability to emulate multiple peers within a single JVM.

5.3.1 Typical Peer Operation Benchmark

The typical operation benchmark measures the latencies of the major peer operations as described in Section 5.1.1. The measurements reflect the time a user waits or an application blocks for the initial response after the operation is invoked. The benchmark is designed so that a single test run includes one invocation of each operation in the sequence that best reflects the typical peer behavior. The JXTA platform is then shut down and restarted for the next measurement run. The chosen order of the operations is required and follows the intuitive and realistic pattern. Moreover, some subsets of the operations need to be performed in the specific order. The platform startup must be the first operation to allow all others to be invoked. Retrieval of the remote pipe advertisement must occur before binding, and retrieval of the local advertisement before the remote one is necessary to ensure the consistence of the local cache.

The operation latencies depend on the chosen platform configuration, which may include rendezvous or relay peers, LAN or WAN deployment, platform invocation from local disk or NFS. All these configuration parameters are external to the benchmark, which runs independently of them and, therefore, needs no adjustment for the specific configuration.

The measurement results reflect the peer group configuration, and the effect of various factors can be observed. Among the many factors that affect the peer operation latencies, this benchmark will show the effect of the size and location of the advertisement cache, the rendezvous and relay peer network distance, as well as the consequent effect on the advertisement discovery and pipe binding.

5.3.2 Pipe Benchmarks

The basic performance metric for a communication protocol or system is the round-trip time (RTT) of the data unit. In JXTA, RTT is measured as the time it takes to send a message and receive the acknowledgement (ACK).

There are several issues with measuring RTT for JXTA messages, which require certain decisions and assumptions be made. The first decision is whether to use ACK or

ECHO messages. Depending on the test payload size, the message and ACK sizes could be significantly different, causing the difference in two trip times. Therefore, half the RTT should not be considered as one-way latency of message transfer.

It is possible to measure the RTT using ECHO message as well. Both payloads would then be the same size, allowing for a more precise estimation of the one-way latency. However, this approach also has drawbacks. Since a JXTA message cannot be reused and simply echoed, it must be re-created resulting in a noticeable overhead. In addition to this complication, the source-based routing adds the routing information to the message in the form of peer IDs, which may be of different size for the original and the echoed message. Moreover, in case of large message sizes, a significantly higher transfer delay is expected.

The ACK approach is used in this study, because it is simpler and faster, as it consistently minimizes the processing at the receiver, and represents a more realistic messaging scenario. A message ID is used to acknowledge the receipt.

Since JXTA pipes are asynchronous and unreliable, polling is used to obtain the ACK as soon as it arrives. JXTA messages may be dropped; therefore, a sensible polling timeout must be used. The sender is not allowed to indefinitely wait for an ACK; rather it assumes the message is dropped if the timeout occurs.

The RTT measurements are taken at the application layer. All overhead costs from converting an XML message to and from a Java object are included in the measurements. These costs are consistent over all send and receive operations. Therefore, this benchmark essentially measures the user’s perception of the message RTT.

The parameters that need to be configured in this benchmark are pipe type, the number and size of message elements. The benchmark is independent of the network configuration, including rendezvous and relay setup.

The throughput benchmark is very similar to the RTT benchmark, except that the sender is not waiting for an ACK; rather it sends the continuous stream of messages. The message ID is used by the receiver to track dropped messages. The sender has an adjustable send rate and burstiness pattern. The independence of the rendezvous and relay setup is maintained; in fact, it is exploited for the relay benchmarking.

5.3.3 Rendezvous Benchmark

The rendezvous benchmark is a modified benchmark from the Bench project [20]. The benchmark can start multiple peers in one JVM, unlike the official JXTA platform. The modifications include changes to the output formatting and the elimination of the password prompt, which do not affect the functionality of the benchmark.

The benchmark starts up the requested number of peers, connects them to the rendezvous and measures the advertisement discovery latency. The parameters include the type of advertisement, query rate, search space and response threshold. The types of advertisements can be pipe or peer. The search space can be larger than the existing advertisement pool to observe the effect of “not found” advertisements. Response threshold is the maximum number of responses to return and the request load is the number of sent queries per unit of time.

5.3.4 Relay Benchmarking Using Pipe Benchmarks

The benchmark suite does not need a separate relay benchmark. Since relays pass messages between regular peers, other messaging benchmarks can be used, such as RTT and throughput pipe benchmarks. The relay use is configured independently from the benchmark setup and the relay configuration is transparent to the benchmark code. The measurement results will reflect the presence of the relay and show its effect on message RTT and throughput.

5.3.5 Additions to the Benchmark Suite

To successfully complete the benchmarking of JXTA following the refined performance model, several additions and modifications to the Benchmark Suite are required. All additions are inside the messaging performance component.

For core pipe messaging, no changes are required for one-to-one and one-to-many RTT and throughput benchmarking. For unicast and secure pipes, a minor change is made to handle multiple senders when measuring pipe throughput.

For non-core services, new benchmarks are added to measure the message RTT and throughput of the bi-directional pipe and JXTA socket. The new benchmarks require new Java classes, because they require separate multi-threaded handlers for incoming messages and connections.

5.4 Summary

This chapter reviewed the evolution of the proposed JXTA Performance Model and Benchmarking Suite, which constitute the first goal of this thesis. The components of the model are discussed and the rationale for the initial inclusion or subsequent addition. The Benchmark Suite components and design issues are also discussed. The presented model and benchmarks are used as the roadmap and tools for the performance evaluation and collection of results.

Chapter 6

Performance Results and Analysis

This chapter presents the performance results collected during this research. The sections are organized according to the four major parts of the JXTA Performance Model. The primary goal of the performance analysis is to observe bottlenecks, performance limits, behavior patterns and effects of different parameters, rather than looking at absolute numbers. To achieve this, an attempt is made to use computers with the same hardware and software as much as possible.

6.1 Testing Environment

Two network environments are used throughout this study, the LAN and WAN. All LAN measurements are taken with peers running on the campus 100 Mbps LAN at the University of Saskatchewan in Saskatoon, Canada. The ping tool measures the average transmission RTT at less than 1 ms on this LAN. The high-bandwidth WAN environment includes peers inside the campus LAN and peers on a home network connected to the Internet using the major Internet Service Provider (ISP) in the same city. Peers referred to as "on the WAN" are located inside the home network connected by a 100 Mbps router, which has access to the Internet via cable modem. The maximum downlink bandwidth of the modem is 4 Mbps and the uplink is limited to 512 kbps. The ping RTT on a WAN recorded averages of 34 to 39 ms at different times throughout the different benchmarking sessions.

The software environment on all peers consists of Microsoft Windows 2000 Professional and JVM 1.4.1.

The hardware environment is a pool of five computers equipped with AMD Athlon 800 MHz CPU and 512 MB of RAM each, located on the campus LAN. Additional computers required for the emulation of multiple peers include one Pentium 4 2.5 GHz desktop PC with 1 GB of RAM on a LAN, and a laptop computer with the same hardware on a WAN. Additional computers on the WAN are used with CPU speeds adjusted to match the campus pool machines by reducing the CPU clock in system settings. Pairs of sender and receiver peers, as well as pairs of rendezvous and relay peers, in both LAN and WAN setups run on comparable hardware.

The JXTA environment assumes the following settings.

The generic JXTA peer group (NetPeerGroup) is tested, but it is constrained to the test peers by disallowing rendezvous or peer connections to the publicly deployed JXTA network. The standard rendezvous query space is set to 1,024 advertisements. The peer group size stated in the results refers to the number of edge peers. The number of rendezvous peers is stated separately and it effectively increases the peer group size. The main underlying transport is TCP.

The results represent the means obtained from the sample size of 10,000 trials with the first 1,000 discarded as a warm-up phase. Smooth traffic is used for throughput measurements, which means a uniform message-sending rate throughout the test run.

If any modifications to the general testing environment or diversions from the stated constraints are made, they are noted with the corresponding results.

6.2 Typical Peer Operations

A peer typically performs a series of operations during its lifetime in a peer network. The measurements of the time required to accomplish these operations were taken with the goal of identifying the cost of each operation relative to others and the startup process. Performance results of the typical peer operations indicate the responsiveness of the platform to the user or application inputs and suggest how to design and implement a P2P system. The relative latencies of the operations indicate those that are preferable and those that should be executed less frequently. Although the order and frequency of the operations is application-specific, it is still worthwhile looking at some typical scenarios.

In a hypothetical scenario, an edge peer performs the given operations once upon startup. When all of the given operations are completed, a peer has joined the peer group, learned about member peers and resources, and decided with which peer(s) it wants to interact. This scenario is in part based on the P2P forum application [11], and in part on the MyJXTA application [33].

The summary results for JXTA 1.0 and 2.0 are shown in Table 6.1. The table shows the mean latency for several peer configurations, using peers in ad-hoc groups and with rendezvous at different network distances. Two edge peers are used for the ad-hoc tests, one is taking measurements and the other provides discovery responses. The additional WAN configuration is using a 56 kbps phone-line modem to connect the second peer to the Internet. The ping tool measured the average RTT as 448 ms for this connection. The route-tracing tool registered 9 hops across the WAN, and between 17 and 26 hops across 56 kbps WAN topology, at different times in this testing session. The peer with the 56 kbps modem is located in Sarajevo, Bosnia-and-Herzegovina, and therefore adds a trans-continental link to the message path. It runs on a 1.33 GHz CPU, 256 MB of RAM and a faster hard disk than in machines on the LAN.

Platform startup: The latency of this operation is affected by the hardware, cache size and the use of a rendezvous or relay. The influence of the hardware is seen in the startup results for a 56 kbps WAN setup, as well as in other results with lower latencies than for LAN setup (group creation, join and retrieval of the cached pipe).

Connecting to the rendezvous/relay introduces an overhead of several socket connections and the exchange of advertisements. Depending on the distance and the network protocol used, connecting to the rendezvous may extend the startup process significantly. In JXTA 1.0, connecting to the rendezvous is more costly, up to 36% or over 1 second. However, in JXTA 2.0 it is barely noticeable, below 20% or under 1 second. The protocol used by the peers has no significant effect on startup, as the difference between TCP and HTTP is not noticeable to the user. More surprisingly, the

Table 6.1: Mean latency of typical peer operations (milliseconds)

Peer Configuration	Start JXTA	Get secure group	Join secure group	Open in-pipe	Get cached peers	Get remote peers	Get cached pipes	Get remote pipe	Open out-pipe	Publish ads	Total
JXTA 1.0											
Ad-hoc	4465.8	1257.5	6.0	16.4	58.8	404.9	25.4	221.2	159.0	80.0	6615.0
Ad-hoc [with 40 ads]	8218.7	1337.2	6.0	20.7	61.8	392.9	28.4	236.1	173.5	81.8	10475.3
TCP Rdv on LAN	5858.3	968.7	6.0	27.0	103.6	726.0	47.6	414.7	248.8	66.5	8400.8
TCP Rdv on WAN	6074.1	1190.1	6.6	16.5	123.3	1639.2	29.8	1173.4	693.2	65.8	10946.3
JXTA 2.0											
Ad-hoc	4822.1	1555.5	5.5	12.8	16.6	448.5	7.1	252.5	708.5	218.9	8047.9
TCP Rdv on LAN	5310.7	1447.4	5.4	10.7	29.3	880.8	2.0	478.7	860.7	294.8	9320.6
HTTP Rdv on LAN	5768.1	1509.0	5.6	12.3	178.0	593.0	2.1	346.5	654.6	307.8	9377.1
TCP Rdv on WAN	5134.5	1429.9	6.0	10.6	27.9	986.5	1.7	557.2	1068.6	310.1	9533.1
HTTP Rdv on WAN	5746.0	1529.9	5.7	12.2	42.2	641.2	1.7	349.7	656.0	307.1	9291.7
TCP Rdv on WAN 56kbps	3475.0	987.6	3.8	130.9	132.1	2485.3	0.2	2303.7	3599.3	296.1	13414.0

network distance matters even less for JXTA 2.0, as shown by the measurements for the LAN and WAN setup with matching protocols.

The effect of the local advertisement cache has several associated issues and it is discussed in more detail at the end of this section.

Joining a group: The process of joining was measured separately for the creation of a group and the actual join operation by invoking the required methods. A peer joins a user-defined secure sub-group (not the NetPeerGroup), using the login name and password. The join is fast because the credentials are checked locally using a peer group advertisement. Note that the group creation involves retrieval of the group advertisement from the cache and creation of the group object, which sets up the environment for the group inside the JVM. For most configurations, this is the second most expensive operation overall and that is a concern for applications in which peers change group membership often and participate in several groups simultaneously.

Discovery of the group resources: The remote discovery of both peer and pipe advertisements (or any other resources) is an order of magnitude slower than the retrieval from local cache (Table 6.1). The results represent the mean time until the first discovery event, which certainly does not mean that this event would give full information about the resources in the peer group. The query for the pipe advertisement specifies the exact pipe, whereas peer query looks for all known peers. Depending on the group size, a peer may wait for a long time to collect the advertisements from all active peers and it can never be sure that all peers responded.

In the case that a peer cannot rely on the cache or discovery, a rendezvous peer should be able to provide accurate information about the peer group, at least in terms of active peers, not necessarily other resources. The cost of retrieving the information from the rendezvous is higher, assuming it is possible to always have one available. A connection to the rendezvous increases startup time, consumes resources by keeping the connections open and the exchange of messages is somewhat slower, since these connections use TCP or HTTP, whereas ad-hoc discovery uses UDP. Placing a rendezvous as close as possible to the edge peers in terms of network distance, ideally on the same LAN, can minimize the discovery cost in some configurations (Table 6.1). However, although it takes more time to retrieve the information from the rendezvous peer, this information is expected to be complete. The overhead is justified if the application requires that peers have updated information about the group.

Opening output pipes: The time it takes to open a pipe can be strongly affected by the network distance between peers and the protocol used. Combined with the remote pipe discovery, it turns out that connecting two peers on a LAN may take in excess of 1.3 seconds; across the Internet several seconds (Table 6.1). Developers should therefore try to reuse the advertisements and open pipe handlers, whenever possible. Contrary to expectations, binding an output pipe over HTTP is faster than over TCP, in both LAN and WAN setups of JXTA 2.0 peers, due to the implementation bug in the platform’s TCP module. The difference is more significant in a high-speed WAN setup, which is exactly the scenario where HTTP is more commonly used. This is the most noticeable difference between JXTA versions, where newer JXTA 2.0 consistently exhibits higher latency for this operation.

Publishing advertisements: The results from Table 6.1 indicate that the last operation in this discussion is certainly not the least important. The low cost of this operation in all configurations does not mean there is nothing to discuss. On the contrary, this operation may have a significant impact on the peer community as a whole.

By publishing the advertisements, a peer really pushes the information about itself closer to the potential users, which is a core concept of P2P computing. Published advertisements are sent to the rendezvous or indexed by it and, if multicast is used, sent to other peers in the group as well. Therefore, by publishing a peer raises the awareness of other peers about the available resources and saves them the cost of discovery. This reduces the discovery traffic bursts and delays at the querying peer. When used properly with tuned advertisement lifetime settings, performing this relatively low-cost operation supports a more efficient peer network.

Publishing advertisements and discovering resources achieves the same goal in different ways. Active discovery by sending queries is at least an order of magnitude slower than retrieving the cached resources. Resource advertisements are cached as XML documents. The search is based on matching the element name-value pairs. However, to use caching, several conditions must be met, *cache access must be efficient*, *the cache must be up to date*, and *all peers must cooperate*.

An *efficient cache* reduces the load on the query receiver, so it can more quickly determine if it has the response, or whether it should forward the query. The major improvement in JXTA 2.0 is the use of the Xindice native XML database [1], which significantly improves the local cache access and management for both edge and rendezvous peers. The improved access to the cache is seen from the results for the retrieval of the cached and remote advertisements in Table 6.1.

However, the *cache must be up to date* to prevent searching through stale advertisements and to avoid responding with inaccurate information. JXTA 1.0 stored the advertisements in separate files, resulting in a lengthy text-based search and potentially excessive disk access during the periodic cleanup process.

A local cache allows a peer to quickly retrieve the desired information without sending remote discovery requests. *All peers must cooperate* for this feature to succeed. Peers should remotely publish their resource advertisements, which is a low-cost operation (Table 6.1). The publishing should be done with carefully set expiry time. The expiry time should match the availability of the resource and the peer's intention to re-

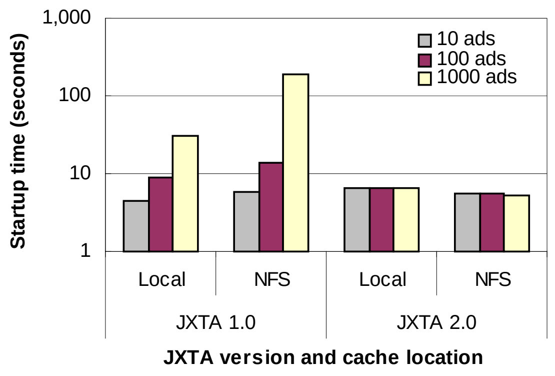


Figure 6.1: Rendezvous peer startup time ([14])

publish. This expiry time may be an approximate session lifetime of the peer, or the scheduled availability of the resource, such as a pipe or service.

The effects of the network distance and local cache size are best seen through the time it takes a 1.0 rendezvous/relay peer to start the platform (Figure 6.1). The startup time is measured for the cache located on an NFS and on local disk. For JXTA 1.0, the values for NFS grow at much higher rate than for local disk. For a 1.0 rendezvous peer starting with 1,000 advertisements, the cost of the NFS is prohibitively high. For example, in a campus environment, where users of computer labs are forced to use the NFS, it may not be desirable to configure a peer to act as a rendezvous if the advertisement traffic is high.

In JXTA 1.0, the cache size has a strong impact on the startup time as well, as seen by the increasing startup times as the cache grows (Figure 6.1). The cache cleanup process significantly extends the startup time. In JXTA 2.0 another significant improvement that came with the Xindice database is the constant startup time (regardless of the cache size). A peer starts as fast with the cache size of 1,000 as it does with 10. Moreover, the effect of the file system location is eliminated, even giving the advantage to the NFS, which in this environment use a highly optimized disk array.

Although local caching is not mandated by the JXTA protocol specification [21], its benefits are obvious when comparing the time required for retrieving locally cached vs. using a remote peer and pipe advertisements. The trade-off comes primarily with the cost of startup. To exploit the cached information, a peer needs to keep the cache updated, which is done during the platform startup.

For applications where peers restart frequently, the overhead of the cache maintenance in JXTA 1.0 may adversely affect the user's experience. It is also possible that all cached advertisements have expired, so clearing the cache is much more efficient than the examination of each advertisement. In highly dynamic applications, such as file sharing, edge peers could be better off to start with an empty cache.

The results from Table 6.1 also show the improvements and regressions between JXTA versions. For example, the opening of the output pipe, which is a costly operation, suggests that the open pipe handles should be reused, especially since this operation takes longer with the newer JXTA version. In addition, the platform startup with the empty cache takes longer in JXTA 2.0, possibly caused by the initialization of Xindice database. There is however a trend in JXTA that newer releases start slower [13], due to modifications and added features.

6.3 Messaging Performance

The JXTA messaging performance is measured using the message RTT and message and data throughput. Message RTT is explored with respect to the pipe type, message size and composition. The throughput benchmarks use different pipe types and two message sizes. The measurements are first obtained in a setup of two directly connected edge peers on the LAN and WAN, then using multiple senders and receivers.

6.3.1 Core Pipe Performance

Pipe RTT is evaluated with five message sizes, which covers most P2P application types, from simple chat applications to file sharing and content storage. The 10 MB message size is removed from the WAN test of JXTA 2.x because of feasibility reasons, as well as imposed limitations in the more recent platform versions.

6.3.1.1 RTT and Message Size in One-to-One Communication

The objective of this test is to understand how different types of pipes behave over the range of message sizes. Each message contains a single payload element

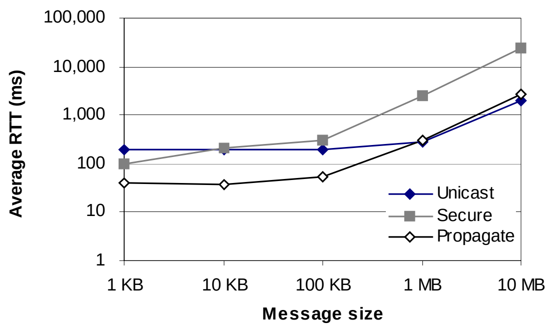


Figure 6.2: RTT of 1.0 pipes ([13])

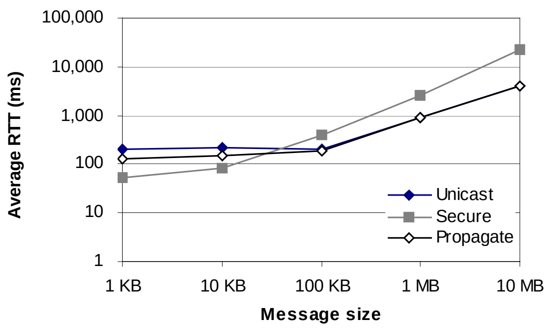


Figure 6.3: RTT of 2.0 pipes ([13])

ranging in size from 1 KB to 10 MB. The messages are exchanged between two peers using a direct TCP connection, or over UDP for propagate pipe on the LAN.

Figure 6.2 shows the average RTT of three types of JXTA 1.0 pipes for each of the message sizes in one-to-one communication (both axes are log scale). Each pipe exhibits a slow, close to linear increase in RTT from 1 KB to 100 KB, even up to 1 MB for the unicast pipes. Between 100 KB and 10 MB, the message RTT over the secure and propagate pipes increases at a different and much higher rate, but it seems to also keep the linear trend. By relative comparison, the secure and propagate pipes differ in an order of magnitude consistently. Unicast pipe performs closer to the secure pipe for smaller messages, but converges with the propagate pipe for larger sizes.

The same benchmarking is performed for JXTA 2.0 and the results are shown in Figure 6.3. The general trends are the same as observed for JXTA 1.0. However, some differences do exist. Unicast pipes perform much slower on JXTA 2.0 for message sizes from 1 MB to 10 MB, up to the factor of two. Secure pipes perform much better on JXTA 2.0 for smaller message sizes (1 KB and 10 KB). This can be explained by the newer and more optimized implementation of secure pipes. Propagate pipes perform significantly slower on JXTA 2.0 over all message sizes, up to the factor of three. In fact, the only improvement of JXTA 2.0 in this benchmark is for small message transfers over secure pipes.

The JXTA 2.0 pipes are next evaluated on the WAN. The average RTT is shown in Figure 6.4. The RTT follows the same patterns as on the LAN, with the expected shift due to the network distance and bandwidth limitations. What is more interesting is that

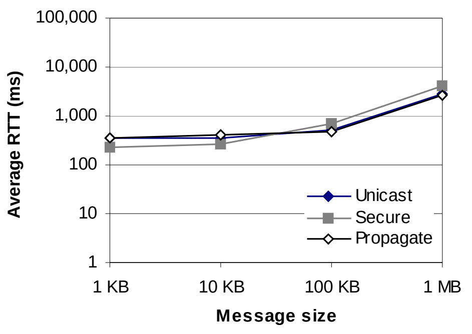


Figure 6.4: Pipe RTT on WAN (2.0)

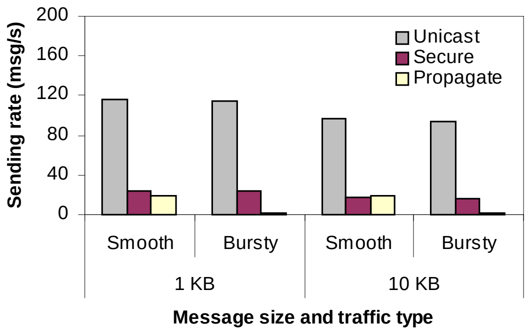


Figure 6.5: Sending rate limits for 1.0 pipes ([14])

all types of pipes have much closer RTT than on the LAN. This means that for wide area deployments there is a smaller penalty in RTT performance with added features, such as security and propagation. A small overhead for an additional feature is a desirable behavior for JXTA pipes.

6.3.1.2 Pipe Throughput in One-to-One Communication

The message load a pipe can sustain is measured by the message and data throughput. Message throughput refers to the maximum number of messages a pipe can reliably transmit, whereas data throughput measures the reliable byte transfer.

The most interesting observation for both JXTA 1.0 and 2.0 is the limited sending rate on all types of pipes. In a LAN configuration of JXTA 1.0 (Figure 6.5), the unicast pipe reached the maximum sending rate at 115 messages per second (msg/s) for 1 KB, and 97 msg/s for 10 KB messages delivered at a smooth rate by the application. Under the same conditions, a secure pipe sends at about 23 msg/s for both 1 KB and 10 KB messages. The unicast and secure pipe implementations achieve the sending rate limits without dropping messages. In addition, the sending queue amortizes the bursty message delivery by the application very well, again without drops. The bursty traffic in the test for unicast and secure pipes consists of 50-message bursts and 500 ms idle time.

For the propagate pipe, the sending rate of about 20 msg/s is both the maximum no-loss rate and the rate that an application should attempt. A higher sending rate causes message drops from the sender's queue. The very low throughput value for bursty traffic a propagate pipe can sustain indicates the failure to accommodate any significant burstiness (Figure 6.5). The most a propagate pipe can handle are 2-message bursts

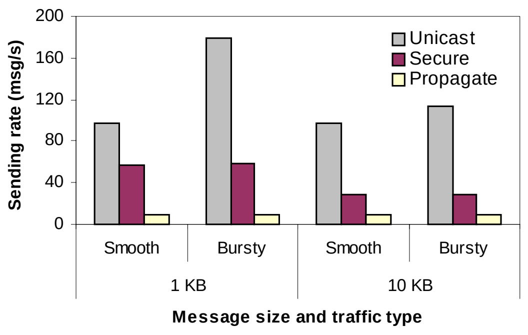


Figure 6.6: Sending rate limits for 2.0 pipes ([14])

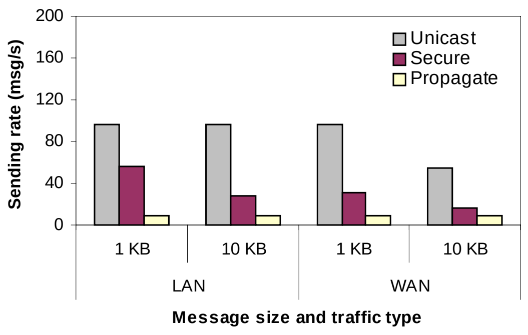


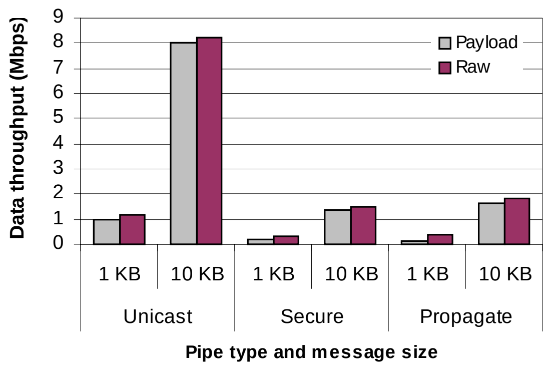
Figure 6.7: Sending rate limits on LAN vs. WAN (2.0)

followed by an idle period to achieve the maximum throughput without drops at the sender, but still has a small drop rate at the receiver.

The JXTA 2.0 implementation shows quite a different behavior (Figure 6.6). The unicast pipe achieves higher throughput for bursty than for smooth traffic (up to 179 msg/s for 1 KB message). Its throughput of smooth traffic is lower than for JXTA 1.0. The secure pipe achieves much higher throughput of both smooth and bursty traffic for two message sizes tested. The propagate pipe is more reliable and it can handle bursty traffic well, but at the price of the lower throughput than in JXTA 1.0.

The effects of the wide-area deployment are investigated next. Figure 6.7 shows the throughput of the 1 and 10 KB smooth message streams on the LAN and WAN side by side. It is noticeable that the WAN configuration has a different effect on different pipes. The throughput of the unicast pipe is reduced by almost a half when the message size is increased to 10 KB. The secure pipe achieves lower throughput (up to 47%) on a WAN for each message size. The propagate pipe, on the other hand, takes the same load equally well regardless of the network distance or message size.

Considering the relatively low throughput in msg/s, it is important to look at what that means for the data throughput of JXTA pipes. As Figure 6.8 shows, larger messages achieve higher data throughput, which is not surprising because the number of messages is the limiting factor, and not their size. The data throughput is obtained by multiplying the message throughput by the message size. It is also interesting to look at the difference between the raw vs. payload throughput. The payload throughput



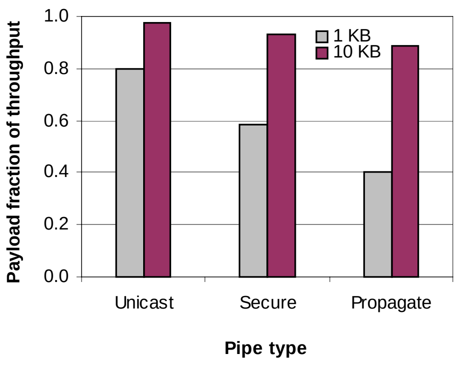


Figure 6.8: Data throughput of 1.0 pipes ([12])

Figure 6.9: Payload fraction of data throughput ([12])

measures the actual useful data payload transmitted in the message, and it is considered an important metric in the performance evaluation of distributed object architectures [19].

The control information in a JXTA message may account for a significant portion of the overall volume of bytes transmitted. This is clearly shown by the impact the extra information has for smaller messages, such as 1 KB, compared to large payloads of 10 KB. Figure 6.9 shows the fractions of the throughput that are actual payload data for pipes connected over TCP. For 10 KB payloads, the overhead data is under 20%, whereas for small messages carrying 1 KB payload, the overhead may account for up to 60% of the message. It is noticeable that propagate pipes have the highest overhead over TCP connection and unicast pipes the lowest.

In JXTA 2.0, there is a significant improvement in this aspect. The XML message control overhead is virtually eliminated in the newer version for the unicast and secure pipes. The overhead on the propagate pipe is considerably reduced as well, by 30% for 1 KB message size.

6.3.1.3 Pipe Scalability with Multiple Receivers

The propagate pipe is already evaluated in a direct one-to-one communication, but its true purpose is to pass messages from one sender to multiple receivers. This is accomplished via a rendezvous peer or by using multicast inside a subnet. Several combinations of transport protocols can be used between the sender, receivers and a

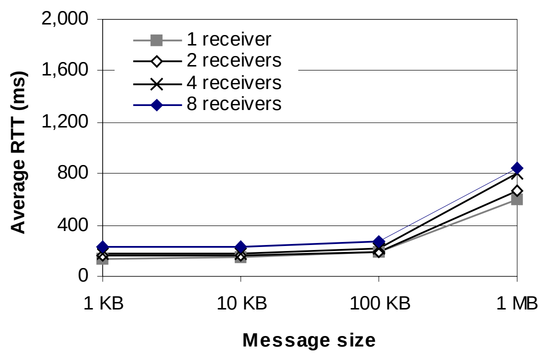


Figure 6.10: Propagating to multiple receivers on LAN (2.0)

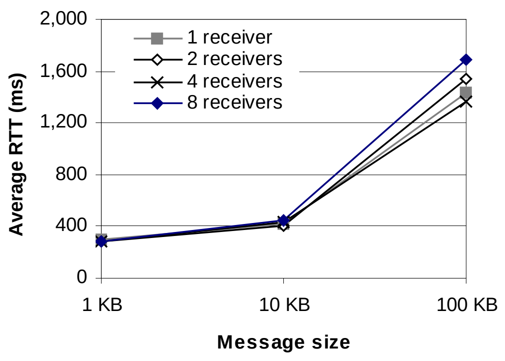


Figure 6.11: Propagating to multiple receivers on WAN (2.0)

rendezvous peer. If the whole peer group is on the LAN, multicast can be used for propagation. In case that multicast is not an option, peers communicate using TCP or HTTP, such that the sender passes a message to the rendezvous peer, which propagates it to all connected receivers. A combination of the two is also possible, where the sender has a TCP or HTTP connection to the rendezvous, which uses multicast to propagate messages to the receivers. This may be required if the sender itself is unable to use multicast for any reason, including when it is outside the subnet.

Figure 6.10 shows the RTT for different massage sizes and numbers of receivers in the latter protocol configuration with TCP. The performance and scaling properties in this configuration are overall better that in the other two. The all-TCP configuration yields slower RTT in all combinations of message sizes and number of receivers. The all-UDP (multicast) setup, on the other hand, performs somewhat better for 1 and 10 KB messages, but noticeably slower for larger (100 KB and 1 MB) messages. Overall, it can be concluded that the propagate pipe performs and scales well with multiple receivers on a LAN.

A similar characterization can be given to the WAN results (Figure 6.11). The actual RTT is higher due to the network distance. More relative difference is also noted between the values for different message sizes, but very little difference when using increasing numbers of receivers. Fairly high RTT values for 100 KB messages are due to the saturation of the lower uplink bandwidth (512 kbps) that was used for the direction of payload transmissions.

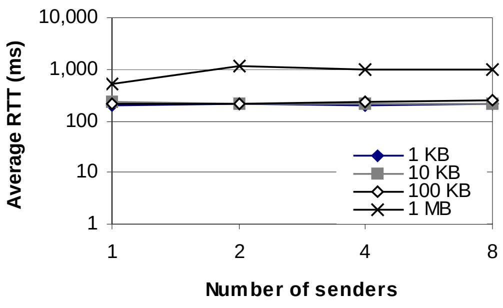


Figure 6.12: Multiple unicast senders on LAN (2.0)

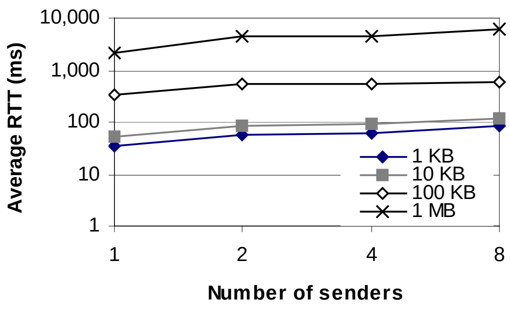


Figure 6.13: Multiple secure senders on LAN (2.0)

The message throughput of the propagate pipe on both the LAN and WAN with multiple receivers (2, 4 and 8) is as good as with a single receiver over a direct connection. The mean throughput of a smooth stream for 1, 2, 4 and 8 receivers shows a very high stability, even across all protocol configurations. The mean throughput is 9.06 msg/s with the standard deviation of only 0.09 for 1 KB messages, and 8.98 msg/s with the standard deviation of only 0.19 for 10 KB messages. Overall, the propagate pipe has remarkably stable throughput performance on LAN and WAN configurations.

6.3.1.4 Pipe Scalability with Multiple Senders

The JXTA unicast and secure pipes can accept simultaneous connections from multiple sending peers. The following results show how the pipes behave with up to 8 parallel senders. The receiver peer runs at 2.5 GHz PC with 1 GB of RAM in this test and the senders run in pairs on 800 MHz PCs. All LAN and WAN results are based on the TCP connections.

Both the unicast and secure pipe show excellent scalability with increasing number of senders across message sizes on both the LAN and WAN. Figure 6.12 shows the average RTT results on the LAN for the unicast pipe, and Figure 6.13 for the secure pipe. The graphs for WAN are very similar, with the actual latencies higher due to the increased network distance. The unicast pipe shows noticeably higher latencies for 100 KB message size than for 1 and 10 KB, unlike on the LAN. The increasing number of senders has very little effect on the performance of pipes, as long as the bandwidth is not saturated (theoretical 4 Mbps on the WAN).

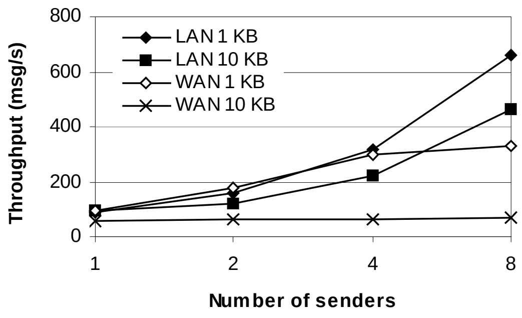


Figure 6.14: Unicast pipe throughput

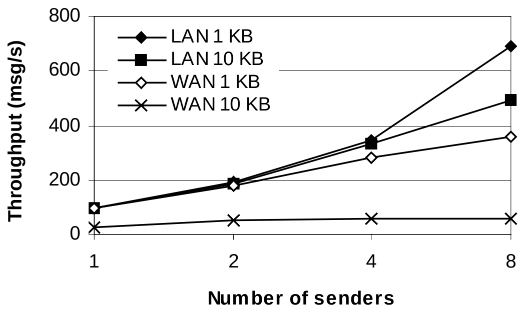


Figure 6.15: Bi-directional pipe throughput

Figure 6.14 shows the throughput of the unicast pipe with increasing number of senders. The value for 1 KB messages from 8 senders does not follow the expected direction and it seems to fall short of the expecting point. The only obvious reason for this would be the bandwidth saturation. The secure pipe shows similar patterns of throughput on both the LAN and WAN setup. Overall, the pipes exhibit an approximately linear increase in throughput rates with the increasing number of senders, which is exactly what the JXTA user would like to see. The unicast and secure pipes accept additional senders at a very low throughput cost up to the bandwidth limit.

Another significant result is that the JXTA receiver (input pipe endpoint) can handle significantly more input than the sender can produce output. The boundaries obtained on the sending and receiving rates will provide the important messaging parameter ranges for the future JXTA simulator.

6.3.2 Non-Core Service Messaging Performance

Bi-directional pipes and JXTA sockets are analyzed as the special non-core messaging services. Both use a different API than core JXTA pipes. The goal of analyzing the non-core messaging services is to find if any trade-offs exist between the features and performance, as compared to the core pipes.

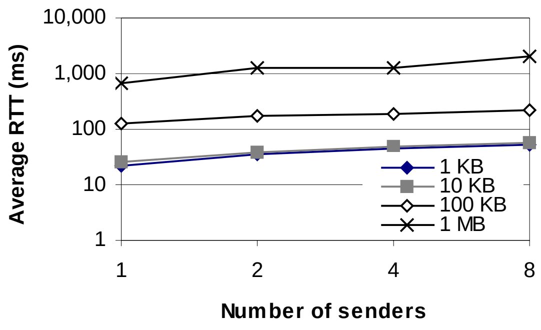
6.3.2.1 Bi-directional Pipe

The performance of the bi-directional pipe is evaluated through messaging on the LAN and WAN, using message sizes of 1, 10 and 100 KB. Larger message sizes are not permitted by the implementation. Furthermore, the limitation is lowered to 64 KB for all pipes in the newest JXTA version (2.3, released in June 2004).

The RTT measurements obtained are very close to the unicast pipe, which is not surprising since the implementation is based on the unicast pipe. On a LAN, the bi-directional pipe yields results that are within 9%, and on a WAN within 15% of the unicast pipe. The RTT performance difference between the unicast and bi-directional pipe is therefore negligible for practical purposes. The throughput results show an interesting difference (Figure 6.15). The bi-directional pipe has a better receiving throughput on a LAN than the unicast pipe, especially noticeable for 2 and 4 senders of 10 KB messages. At the highest load with 8 senders and 10 KB message size, the total byte throughput with JXTA messaging overhead, exceeds 65% of the network interface and LAN bandwidth. This leads to the conclusion that JXTA can effectively use the available bandwidth and that the message receiving capacity is very good.

On the WAN, there are several differences in performance. The throughput of 1 KB messages is within 9% of the unicast pipe, but this is a situation of bandwidth under-utilization. For 10 KB messages 2 senders already saturate the line capacity. The bi-directional pipe has a lower throughput than the unicast pipe, especially for a single sender at only about half of the unicast rate. Moreover, the reliability of the WAN connection is extremely low, reflected by a high message loss (message queue overflows) if the sending rate is higher than the maximal sustainable at the receiver.

The trade-offs between the unicast and bi-directional pipes exist primarily in the reliability vs. programming preference. Although a bi-directional pipe has a more optimized implementation, its performance is very close to the unicast pipe in most cases. Its major advantages are the single messaging object for bi-directional communication and the API similar to the classic socket programming of Java. The main disadvantages are low reliability and the need to implement a custom multi-threaded server. However, the bi-directional pipe will be reliable in JXTA 2.3 and the multi-threaded implementation is close to trivial for an intermediate Java programmer. Overall, the bi-directional pipe looks like a promising addition to JXTA.



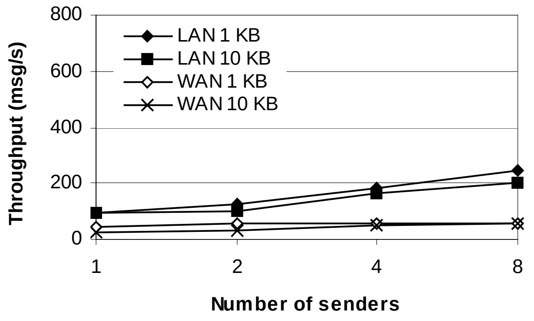


Figure 6.16: JXTA socket RTT

Figure 6.17: JXTA socket throughput

6.3.2.2 JXTA Socket

While the reliability and API are the major advantages of the JXTA socket, the need to implement a custom multi-threaded receiver (server) component is a drawback. The JXTA socket benchmarks use a pair of character strings in place of genuine JXTA messages, but they will still be referred to as "messages". The first string contains the message ID, and the second stores the payload of desired size. The ACK is composed of the single string containing the message ID.

The JXTA socket message RTT is shown in Figure 6.16. For message sizes of 10 KB and less, the JXTA socket is an order of magnitude faster than the unicast pipe, while maintaining very good scalability properties with the increasing numbers of senders. RTT for 100 KB messages is comparable, but still faster than for unicast pipe, while for 1 MB message sizes the result is quite different. JXTA socket is slower than unicast pipe, up to 51% for 8 senders. Therefore, the advantage is clear for smaller message sizes. The possible explanation for the different behavior for small and large messages could be found in the implementation of the reliability provisions. The lack of design documentation, an attribute of many open-source projects, makes it difficult to definitively answer many performance questions without inspecting the source code, which in cases like this, is not feasible.

The performance on the WAN has a similar pattern, again compared to the unicast pipe. The difference is not as significant as on the LAN, where the unicast pipe lags by 25% to 180%, but the situation changes when the amount of sent data approaches the bandwidth limit (100 KB messages and 4 senders). JXTA socket will

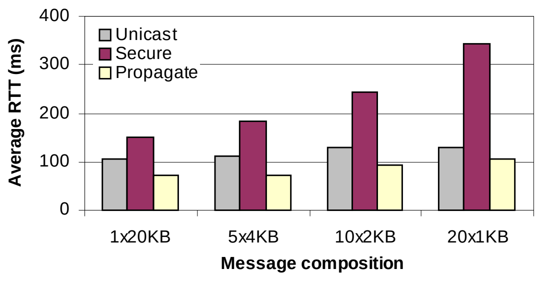


Figure 6.18: Effect of message compositions in 1.0 ([13])

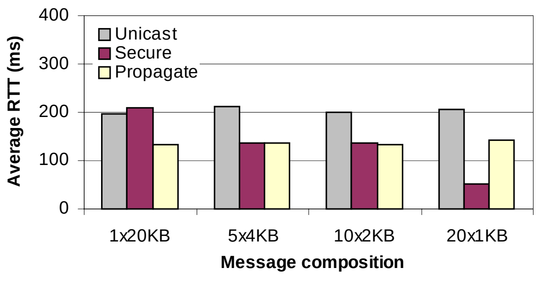


Figure 6.19: Effect of message compositions in 2.0 ([13])

then become slower up to the point of shutting down the cable modem at 8 senders attempting 1 MB messages.

The message throughput measurements give an opposite impression of the JXTA socket performance (Figure 6.17). For both LAN and WAN, and for both message sizes tested, JXTA socket performs worse than the unicast and bi-directional pipes. JXTA sockets seem to have difficulty handling the multiple senders and the possible causes are the multi-threaded socket server and reliability provisions. The single point of "contention" among connection handlers is the synchronized Java method where every incoming message ID is processed. However, this method is common for all pipe benchmarks, including the bi-directional, and may not be the main cause of lower throughput. Although the socket throughput starts as good as for the bi-directional pipe for 1 KB messages with a single sender, it does not keep up for any other configuration. Overall, the performance of the JXTA socket is good for bi-directional message exchange under low load, but not as good for high message loads. The price of reliability is the inferior performance, regardless of the actual reason.

6.3.3 RTT and Message Composition

The following test looks at the overhead of processing the messages composed of different numbers and sizes of elements. Four variations of a 20 KB message are used: 1x20 KB (1 element of size 20 KB), 5x4 KB, 10x2 KB and 20x1 KB.

Figure 6.18 shows the impact of message composition on the performance of JXTA 1.0. As the number of elements increases from 1 to 20, while their size decreases, the average RTT on a secure pipe more than doubles. For a unicast pipe, the increase in RTT over four combinations is 24% and for the propagate pipe about 43%.

For JXTA 2.0 a very different behavior is observed (Figure 6.19). The message compositions have virtually no effect on the performance of unicast and propagate pipes. This is a good property, showing an improvement in JXTA 2.0. However, the message composition effect on secure pipes is different from JXTA 1.0. The newer version handles more small elements better than fewer large ones. Another interesting result is that JXTA 2.0 secure pipes can outperform unicast pipes for smaller message sizes with complex structure. This observation is consistent with the previous RTT test over different message sizes.

The overall effect of message composition is not very surprising, considering that messages are XML documents and the complexity of document structure affects the processing time. This implies that parsing and encryption can significantly impact the overall performance of the JXTA messaging sub-systems.

6.4 Rendezvous Performance

As the essential components of the JXTA framework, a rendezvous peer, and as of lately a rendezvous network, deserve a very close look, because they may attract large amounts of traffic and become a bottleneck on both the peer and physical network. This section reviews performance of a single rendezvous peer, as well as the rendezvous network, in peer groups of various sizes, topologies and query loads.

In the experiments no more than 8 edge peers run on a 2.5 GHz PC. The rest of the peers are distributed over several 800 MHz machines. Rendezvous peers run on separate 800 MHz machines.

6.4.1 Query-Handling Performance and Scalability

All three existing models for resource search and discovery are investigated and compared: centralized, distributed with query flooding, and distributed with SRDI.

In the centralized model, all advertisements are located on the rendezvous peer. Advertisements used for the distributed search tests are evenly distributed across all

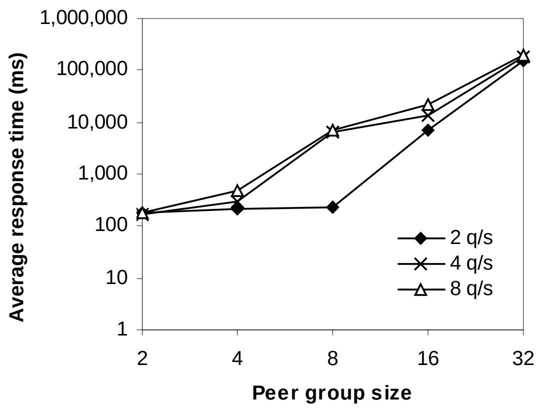


Figure 6.20: JXTA 1.0 centralized
search with 100 ads ([14])

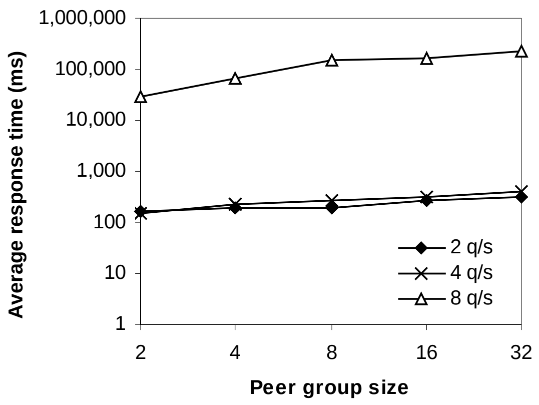
edge peers. Pre-generated peer advertisements are used for all experiments. The search query consists of the string representing the exact peer name that is matched to the name in the advertisements. Peer names are in the form "peer*N*", where *N* is the sequentially generated number. The name for each query is randomly generated from the search space corresponding to the number of pre-generated advertisements.

6.4.1.1 Performance of Search Models

The object location model determines the search performance of a distributed system, but in JXTA, an additional effect is introduced by the underlying content management (CM) system. CM in JXTA 1.0 uses an individual file per advertisement, whereas JXTA 2.x switched to the Xindice database. The main performance metric for a JXTA rendezvous peer is the average response time to the issued query (query response time). This metric is explored in some detail in respect to several parameters; most importantly, the peer group size, query rate and the rendezvous cache size.

The centralized search model is analyzed first, using the measurements on a JXTA 1.0 rendezvous with 100 advertisements and a JXTA 2.0 rendezvous with 100 and 1,000 advertisements. The query response time is presented together with the sending rate, representing the approximate throughput of the rendezvous peer.

Figure 6.20 shows the scaling of the query response time across several peer group sizes and three aggregate query rates at the rendezvous: 2, 4 and 8 queries per second (q/s), on a JXTA 1.0 rendezvous peer caching 100 advertisements. Both the



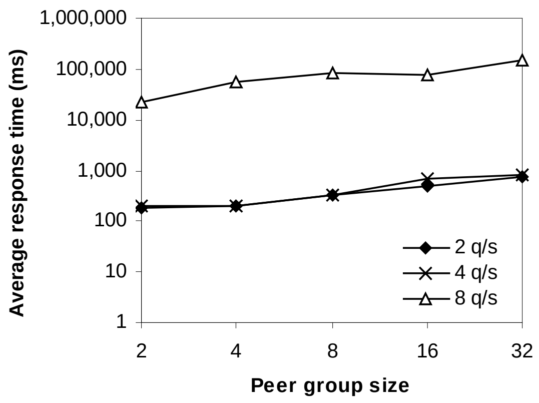


Figure 6.21: JXTA 2.0 centralized search with 100 ads ([14])

Figure 6.22: JXTA 2.0 centralized search with 1,000 ads ([14])

query rate and peer group size affect the response time. The response time increased as much as 50-fold between group size 4 and 16, indicating poor scalability. The JXTA 1.0 rendezvous implementation stores each advertisement in its own file, and the code is not optimized for high performance.

Figure 6.21 presents the average response times of the JXTA 2.0 rendezvous with the cache size of 100. The results indicate that the group size does not matter as much as the query rate. Similar results for the cache size of 1,000 are shown in Figure 6.22. In most cases, the increase in average response time between group sizes 8 and 32 is less than twofold for the same query rate. A JXTA 2.0 rendezvous shows much better scalability properties with small increases in response times over peer group sizes. The most effective factor in improved scalability properties of the rendezvous peers is the Xindice-based cache implementation.

However, there is a downside to the newer JXTA implementation. The 2.0 rendezvous on the aforementioned hardware cannot provide responses in reasonable time at the rate of 8 q/s, whereas 1.0 rendezvous can for small groups. The response times for rates of 2 and 4 q/s are all below 1 second, suggesting that the JXTA 2.0 rendezvous can serve the time-sensitive or user-driven applications at these query rates. On the other hand, the rate of 8 q/s takes the response time into tens and hundreds of seconds, causing the rendezvous to be appropriate only for applications where fast discovery is not important. Therefore, the performance improvements in JXTA 2.0

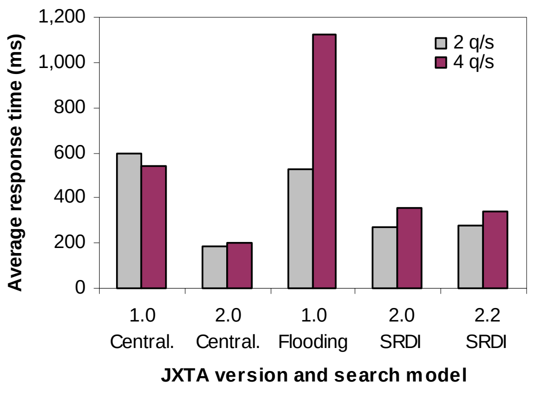


Figure 6.23: Search and discovery models

apply only to lower query rates of 2 and 4 q/s, whereas the general scalability properties seem to extend to the higher rates as well.

The distributed search models are compared next in the similar manner. Figure 6.23 is representative of the performance relationships between the search models, using a group of 4 peers and 1 rendezvous with the query space of 1,024 advertisements. Results for other group sizes follow the same pattern.

The first observation is the superiority of centralized discovery in JXTA 2.0 vs. JXTA 1.0. The significant improvement in response times can certainly be attributed to the better CM system.

The next result relates to the inefficiency of the flooding approach. For a low query rate, there is no particular advantage to using either centralized or flooding search. For a higher query rate, it is usually more desirable to avoid the centralized model because of overloading, but in JXTA 1.0 it is the opposite. The centralized model performs steady at 4 q/s, but the flooding approach slows down noticeably.

The CM system implementation in JXTA 1.0 causes several unintuitive results, due to the local cache cleanup that executes every 30 minutes taking at least 30 seconds. The clean up method looks at all advertisements to find and remove the expired ones. The query responses stop for the duration of the cleanup, which takes longer as the cache grows. The cleanup process dramatically affects about 4.3% of the queries in a sample of 10,000 at a rate of 2 q/s, which causes the mean to significantly increase. This

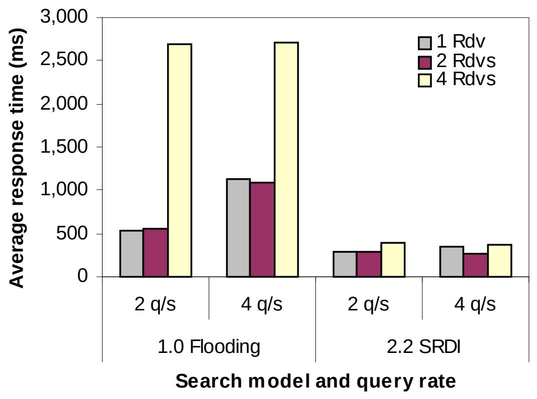


Figure 6.24: Search models and multiple rendezvous

is the reason why the mean RTT for 2 q/s is higher than for 4 q/s for the 1.0 centralized model. The session required to collect 10,000 samples is twice as long under the 2 q/s load, so it suffers from more cache cleanup cycles. The same reason applies for the fact that 1.0 flooding tested slightly faster than centralized for 2 q/s load.

A better solution for distributed search comes with the SRDI in JXTA 2.x, which outperforms the flooding and centralized models of JXTA 1.0. It is also noted that SRDI did not outperform the centralized search of JXTA 2.0, but that is hardly a surprise at these query rates. The benefits of SRDI will become obvious through the discussion on reliability and high load performance.

6.4.1.2 Multiple Rendezvous Peers

The distributed search models are next compared in different topologies; including 1, 2 and 4 rendezvous with edge peers evenly distributed to the rendezvous peers. It is already known that SRDI is superior to flooding, but it is interesting to see if and how multiple rendezvous help the search models. Figure 6.24 shows that in a peer group with 4 edge peers, multiple rendezvous not only fail to help the flooding model, but they actually degrade the performance. This result can be attributed to the very nature of flooding; each rendezvous increases the number of queries in the peer group, further overloading the peers. Due to the exponential increase in the total traffic, the performance suffers massively for a 4-rendezvous configuration already, since the amount of incoming queries is exposing the problems with the incoming query management components of JXTA.

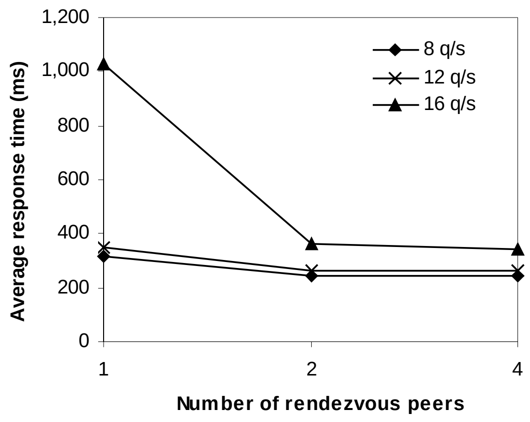


Figure 6.25: SRDI with multiple rendezvous

On the other hand, the SRDI model performs well with multiple rendezvous, proving that it is actually designed for it. For low query loads, it seems nearly irrelevant how many rendezvous are deployed in a peer group using SRDI. A very small increase in response times from 2 to 4 rendezvous peers shows that only a small overhead is added with more complex topology. Adding rendezvous peers for redundancy and load balancing in this case comes at a very low price.

Although the observed benefits of SRDI already make it look superior to other search models, even more is expected from it. The next experiment explores the handling of high query loads by the SRDI, with multiple rendezvous peers. High query loads in this test are those loads at which a single rendezvous starts to drop queries and response times noticeably increase. The effect observed is presented in Figure 6.25. At loads above 8 q/s, adding a rendezvous to balance load can significantly improve response times, especially at 16 q/s. Increasing to 4 rendezvous further improves response times, albeit slightly, but this behavior is opposite to the one observed for lower loads, where 4 rendezvous peers added overhead. The reason why 16 q/s presents a difficulty for a single rendezvous lies again in the processing overhead of the incoming messages.

Overall, two results are worth noting about multiple rendezvous peers. The first is that the additional rendezvous peers in the SRDI model do not hurt performance like in a flooding model (Figure 6.24). The second result is that adding one rendezvous at a

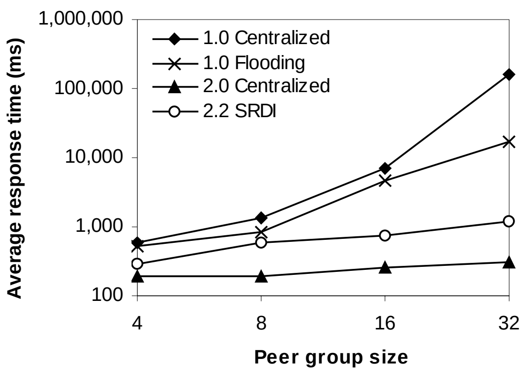


Figure 6.26: Scalability of search models

high query rate appears to be optimal (provides the highest improvement in performance). This point is at 16 q/s for the peer group, hardware and the environment in this study, but it would likely change from system to system. Nevertheless, this result provides a part of the answer to the question about the number of rendezvous needed for a peer group. It also contributes to the scalability picture of the JXTA rendezvous network. The key problem of query handling performance lies in the connection and thread management, as well as processing of the queries within the endpoint service.

6.4.1.3 Scalability with Group Size

The scalability of the search models is further compared in respect to the increasing of the peer group sizes. Figure 6.26 shows the response times in groups of 4 to 32 peers generating a workload of 2 q/s. As expected, performance degrades quickly in peer groups that use JXTA 1.0 flooding or the centralized model. For a two-fold increase in a group size, response times increase as much as ten-fold for the centralized model. Scalability in JXTA 2.x groups is much better. Query response time rises slowly in a peer group that uses SRDI, showing excellent scalability in respect to peer group size. SRDI, therefore, is clearly more scalable than flooding. A similar behavior is seen with JXTA 2.0 centralized model as compared to JXTA 1.0, testifying to better scalability of Xindice-based CM system vs. file-based.

6.4.1.4 Performance on WAN vs. LAN

The measurements are taken on the same overlay topologies deployed on the underlying physical LAN and WAN, to observe if and what differences exist in the

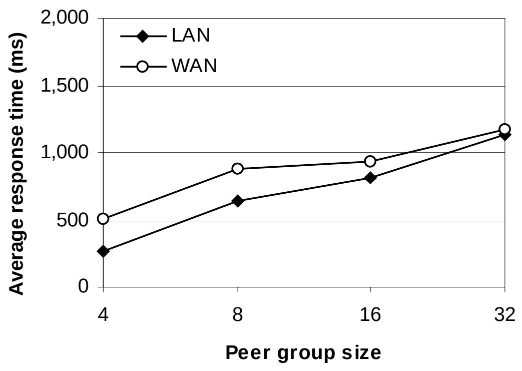


Figure 6.27: SRDI rendezvous response time on LAN vs. WAN

rendezvous performance. The WAN topology in these group sizes places all edge peers in one network and 2 rendezvous peers in another up to the peer group size of 16, whereas in the group of 32, both the edge and rendezvous peers are split into two networks and cross connected. The mean query response times are shown in Figure 6.27 for a query load of 4 q/s and 2 rendezvous peers. The first observation is that the performance on a LAN is better. It was expected that the response times for a WAN topology are somewhat higher, exactly as seen. However, what is more important is that the relative difference between response times on the WAN vs. LAN is getting smaller with the increasing group size (up to 32 edge peers). The result is consistent across different query loads and quantities of rendezvous peers. This means that the physical network distance is less significant as the peer group size grows when broadband connections are available.

6.4.1.5 Positive vs. Negative Queries

Edge peers may search for existing or non-existing advertisements. The standard test setup includes generation of positive queries, which are guaranteed to exist in the rendezvous cache or in the peer group. In a more realistic situation a rendezvous would also face queries that cannot be satisfied, also called negative queries. To test the impact of negative queries on the response time for positive ones, the two types of queries are generated in approximately equal amounts and the response times recorded.

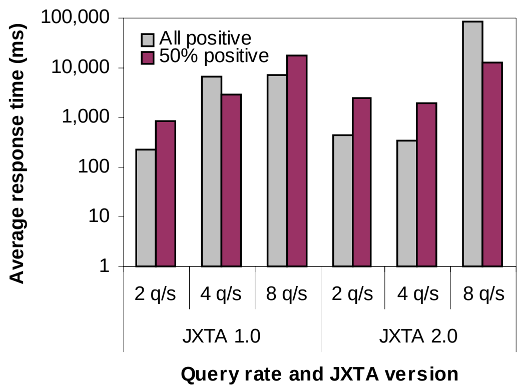


Figure 6.28: Effect of negative queries on centralized search

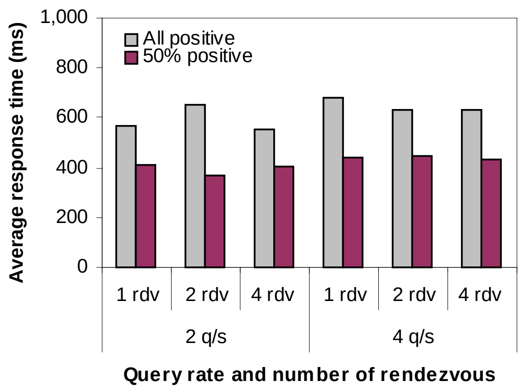


Figure 6.29: Effect of negative queries on SRDI

The comparison of response rates between the positive and mixed query workloads is presented in Figure 6.28 for the centralized search model. In most cases, faster responses are recorded for positive query workload, meaning that searching for non-existing advertisements is costly and affects the response time for the successful search. Similar behavior is observed for both JXTA versions and different cache sizes. However, this result cannot be generalized, because the proportion of positive vs. negative queries may vary significantly between applications.

The opposite effect is consistently observed for SRDI, where negative queries in fact reduce the load on the rendezvous peers, as seen in Figure 6.29. Although the negative queries cause more traffic by activating the limited-range walker, this surprisingly does not hurt the performance. It can be concluded that the walker traffic is not nearly as costly as the response traffic, which is a favorable result for the concept of a loosely consistent network.

6.4.2 Resource Usage

The resource usage of the rendezvous peer is another important metric. It reveals the need for a dedicated machine and may affect the overall peer network performance. The CPU usage is measured and it represents the total processor time that the rendezvous process consumed and memory allocation represents the portion of the RAM used.

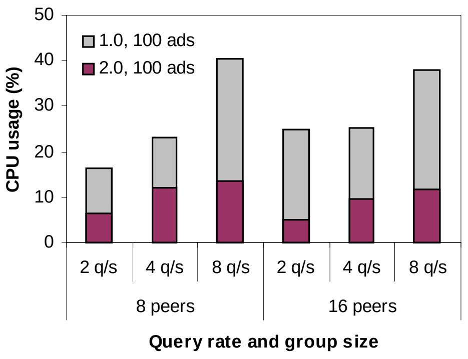


Figure 6.30: Rendezvous CPU usage ([14])

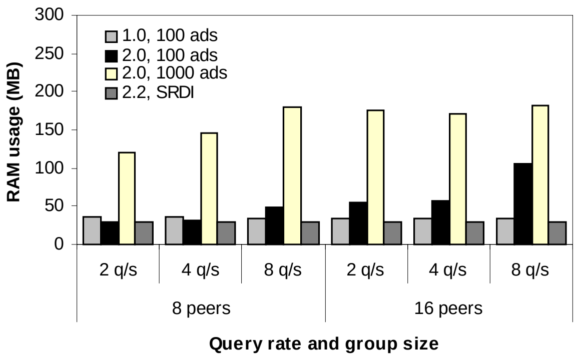


Figure 6.31: Rendezvous RAM allocation

For centralized search, where advertisements are located at the rendezvous peer, the CPU usage of the JXTA 2.0 implementation is at most half of the 1.0 implementation across the tests. This reduced CPU consumption can be viewed as the major improvement in JXTA 2.0. The actual values for JXTA 2.0 do not go above 14%, exhibiting almost negligible impact on the CPU (Figure 6.30). In addition, the rate of increase across query rates seems smaller than for JXTA 1.0. Another favorable result for JXTA 2.0 is that the average CPU usage does not increase with group size for the same query rate. The CPU usage is also independent of the cache size, since the results for JXTA 2.0 with the cache sizes of 100 and 1,000 are almost identical.

Overall, the CPU usage results suggest that it is not necessary to invest in high processing power to afford a rendezvous for peer group sizes of up to 32. This is extremely important for the deployment of JXTA solutions for home users, since it allows for the creation and support of new peer groups without high-powered machines.

The JXTA 1.0 rendezvous is more conservative in memory allocation than 2.0 rendezvous is (Figure 6.31). For JXTA 2.0 memory allocation for the cache size of 1,000 is higher than for 100 as expected, but it is excessive considering the actual volume of data accessed and processed. In fact, most of the heap allocated by the JVM is not released after the initial allocation, so the memory consumption in this case is not really a JXTA issue. Nevertheless, without implementing a custom garbage collection schedule, the obtained measurements of memory allocation reflect the default requirements.

The CPU usage and memory consumption results present a significant trade-off in case of the centralized search. However, when SRDI is used, memory consumption falls drastically, while keeping the CPU usage at the same levels, which is in fact the most desirable option. All SRDI indices and data structures are maintained in memory, so there is a potential for even lower memory requirements should the disk-based caching be implemented. Long-term performance of SRDI still raises some questions, since the memory consumption rises with time. At 8 q/s in a group of 16 peers, a rendezvous using SRDI allocates 45 MB of RAM in 12 hours and 85 MB in 20 hours of running. This may require occasional restarts of the rendezvous peer, depending on the available RAM and query load. The memory leak in the SRDI index causing this behavior is fixed for the next release of the reference implementation.

6.4.3 Reliability

The number of dropped queries determines the reliability of a rendezvous peer. Queries are dropped by the rendezvous when the message queue overflows or when a query cannot be satisfied. If the edge peers generate queries that a rendezvous can satisfy, such as in these tests, the dropped messages in fact reflect the queue overflows. The rendezvous service protocol is not reliable, so dropped queries should be expected depending on the workload. Other metrics of reliability are the variation in response times and the success rate of failure recovery. All of these metrics are evaluated and discussed.

The dropped query rates for the centralized search are shown in Figure 6.32 for group sizes 8 and 16. The results do not favor any of the JXTA versions. My impression is that very high fractions of queries are dropped especially as the query rate increases. This suggests that there are serious queue management issues in both JXTA versions. However, there is no indication that the cache size adversely affects the drop rate, as results are similar for cache sizes of 100 and 1,000. The drop rates for flooding are even higher than for the centralized search, reaching over 80% at higher loads. A major improvement is noted for SRDI, which recorded either none or negligible number of dropped queries at loads up to 8 q/s. This is why SRDI is preferable to the centralized

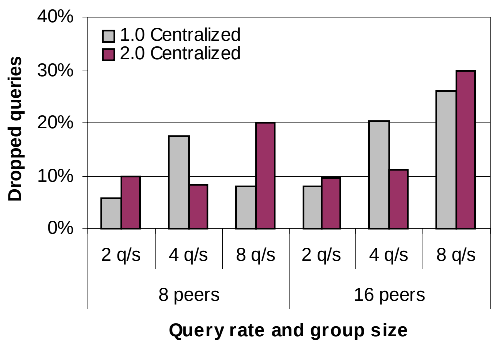


Figure 6.32: Query loss for centralized search ([14])

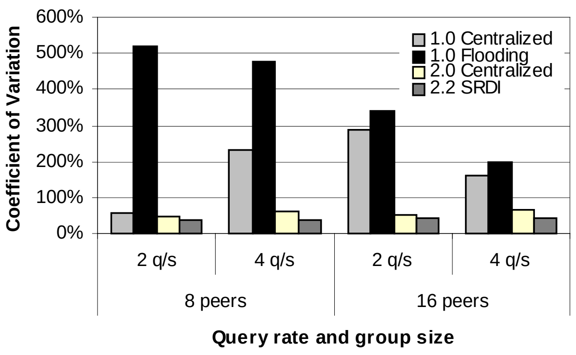


Figure 6.33: Coefficient of variation for search models

model in JXTA 2.x, even though the centralized search yields faster responses at loads below 8 q/s (Figure 6.26).

Response time variation is another component of rendezvous reliability. The average response time gives a high-level picture and helps characterize the performance and scalability, but it is necessary to consider the likelihood of a response to a query arriving within a certain time. The insight into this issue is given by the coefficient of variation (CV). CV is calculated as a ratio of the standard deviation and the mean for different peer group sizes, query rates and search models.

The results are shown in Figure 6.33. Overall, the variation in response times is very high, mostly around and above 50%. It is encouraging to see that the newer JXTA versions bring the variability down in most cases and that the differences in CV across query rates and peer groups are smaller. The extremely high CV for JXTA 1.0 indicates that the developers and users could hardly make any prediction about their rendezvous peer and its response times. High variation also means unfairness to some users, who face a dilemma whether to wait or resend the query. This adversely affects user's experience and the performance of the system due to the potential generation of unnecessarily repeated queries. In addition, automated systems are harder to program; in particular it is difficult to determine a good value for resend timeouts. SRDI exhibits the most reliable and predictable performance, which further reinforces it as a search model of choice for JXTA applications.

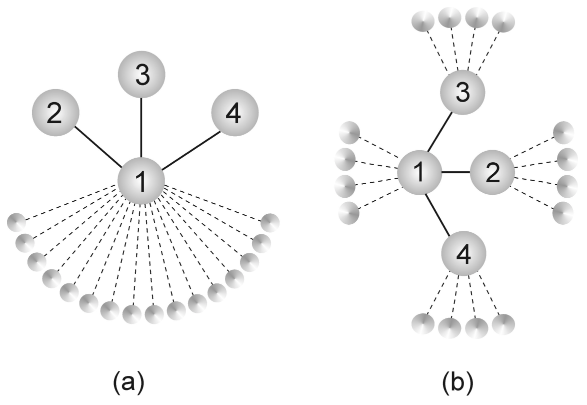


Figure 6.34: Initial rendezvous seeding

The evaluation of the JXTA rendezvous network would not be complete without testing the recovery after failure. The effect of the rendezvous failure depends on the ability of the rendezvous network to repair the SRDI and the ability of the edge peers to reconnect to one of the remaining rendezvous.

The peer group for this test consists of 16 peers and 4 rendezvous in an ad-hoc topology construction. Five test runs are executed, each with similar results. The peer group topology was not predefined and peers were allowed to connect to any of the known rendezvous. Therefore, the initial configurations were not symmetric, with the number of peer connections per rendezvous ranging from 2 to 9 across the test runs. At 10-minute intervals, one rendezvous peer was removed, which left the peer group with three, two and finally one rendezvous at the end. The rendezvous peers are removed by killing the process, so the edge peers cannot be notified of the rendezvous departure.

The following behavior was observed during the test runs. After the first and second rendezvous are removed, all edge peers recovered by connecting to another rendezvous. Only after the third failure, with only one remaining rendezvous in the group, some peers never recovered.

In respect to the number of peers that never recovered, it seems that it depends on several complex factors. The first is the initial rendezvous configuration. If all peers know about one and the same (seed) rendezvous at startup (Figure 6.34a), they are initially distributed among other three rendezvous, and the seed rendezvous remains to help in recovery. In this case, the seed rendezvous is used as a last resort when peers cannot connect to any other rendezvous they learned about in the meantime. All peers ultimately recover by connecting to the seed rendezvous at the end. Another factor is the seed rendezvous failure. If the seed fails, then earlier it fails, more peers end up disconnected at the end. Since the peers are seeded with only one rendezvous peer, it is unlikely that they will be able to discover all of the others.

The initial network may be connected differently. Peers may be seeded with one rendezvous at startup, but the seeds are different for the edge peers (Figure 6.34b). The different seed rendezvous again know about the same one rendezvous. Now multiple rendezvous peers act as seeds, which may not be desirable. In this case, all rendezvous peers start with at least one edge peer connection. This configuration also ends up with peers that fail to recover after three rendezvous failures. These observations suggest that edge peers should use one seed rendezvous to join the network and thereby have the highest likelihood of recovery after the rendezvous failure. However, a more detailed investigation of rendezvous failure recovery would give a more definitive answer on all reasons for recovery behavior.

Queries are dropped during the period of disconnection, but after the peers recover, search and discovery works normally, without loss at the recovered peers. The only loss that may still persist is due to the peers that never recovered. This indicates that SRDI repairs itself well.

A pattern of the edge peers’ recovery time also emerges from the conducted tests. Out of 61 disconnected peers during all test runs, 46 recoveries are recorded. In 29 recoveries caused by less than 8 disconnected peers per failed rendezvous, the mean recovery time is 78.4 seconds with standard deviation of 2.2 seconds. The remaining 17 recoveries occurred in 100.4 seconds on average with standard deviation of 5.5 seconds. This suggests that the recovery time depends on the number of disconnected peers, but that a noticeable difference exists with more than 8 disconnections. However, the basic component of recovery time is a minimum wait of 60 seconds before a repair will be attempted. This is due to the configuration of the link failure detection mechanism, which ignores shorter breaks to mask the transient network glitches. Therefore, the actual recovery process took 18.4 and 40.4 seconds on average, for fewer than 8 and more than 8 disconnected peers, respectively.

Overall, the recovery properties of both the rendezvous network and edge peers are good. SRDI rebuilds well, but there may be some room for the improvement of the edge peers' recovery times, depending on the application requirements.

6.5 Relay Performance

Relays are necessary to connect the pipe ends between peers that are not directly accessible to each other. Relays potentially introduce an overhead of communication by adding the processing cost and extending the pipe length. The RTT test measures the cost of relay for a two-way communication. The relay throughput test measures the receiving rate of messages, given some sending rate and the message path through relay. Lower receiving rates can quantify the overhead a relay imposes on a one-way pipe throughput. The messages in this test contain one payload element 1 KB in size. Edge peers communicate in a one-to-one fashion. Four peer and relay configurations are tested:

1. Direct TCP/UDP without a relay,
2. Single relay using only TCP or both TCP and HTTP,
3. Single relay using only HTTP and
4. Two relays using at least two HTTP connections.

Since HTTP is a higher-level protocol that uses TCP, it should introduce additional overhead. Configurations 2 and 4 use both TCP and HTTP for JXTA 1.0, with the goal of measuring the effect of enabling TCP although relaying functionality uses only HTTP. For JXTA 2.x configuration 2 uses only TCP and configuration 4 has two HTTP connections (between the sender and its relay and between two relays) and one TCP connection (between the receiver and its relay). The addition of the TCP in configuration 4 is to prevent the edge peers from optimizing the path by converging to the single relay, as provided by the newer implementation. The RTT results for JXTA 1.0 on a LAN are shown in Figure 6.35 and for JXTA 2.2 on a WAN in Figure 6.36.

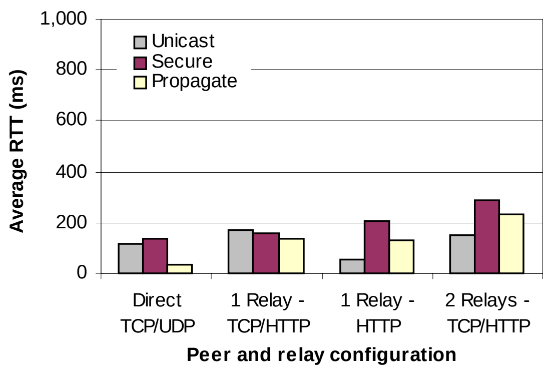


Figure 6.35: RTT through 1.0 relays ([14])

The figures are representative of the effects that relays have and encompass most scenarios. The results for other combinations of JXTA versions and network configurations follow the same patterns.

6.5.1 Effects of Relays and Transports on Message RTT

In the direct TCP/UDP configuration, two peers connect directly without any rendezvous or relay. This scenario is shown as a baseline for easier comparison with relayed configurations. Propagate pipes show the best performance on a LAN because the peers use the available IP multicast.

In all of the relayed configurations, the core pipe performance trends are consistent, but more emphasized on a WAN. The most intriguing is the effect of the relay and transport on the unicast and propagate pipes. They perform much better through an HTTP relay than in any other configuration, except when the propagate pipe uses multicast. This is quite an unexpected but consistent behavior, and the reason is found in an implementation bug within TCP transport module that persisted throughout the JXTA versions. The same problem affected other results as well, such as pipe binding. This behavior is reportedly fixed in the next JXTA release (2.3).

Secure pipe becomes uniformly slower through the configurations with added relays, all JXTA versions and both LAN and WAN configurations, as expected.

In the two-relay configuration, the relays pass messages between each other on behalf of their attached edge peers. The additional relay has the strongest effect on the

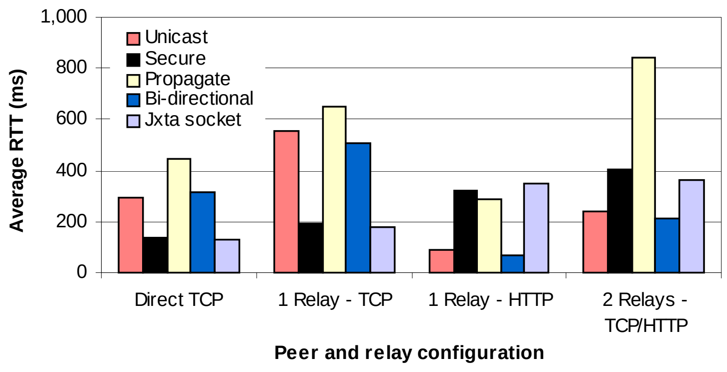


Figure 6.36: RTT through 2.2 relays

propagate pipe causing the RTT to reach the level that will likely be noticeable to the user.

Another interesting result is that performance of the relayed secure pipe using TCP changed between JXTA versions, relative to the unicast and propagate pipe. The improvement is seen in the first two configurations. It is also surprising to see that the secure pipe outperforms the propagate pipe in the two-relay configuration. It was expected that the security provisions would constitute an additional overhead.

For JXTA 2.2 the bi-directional pipe and JXTA socket are added to the analysis with the following observations. Both non-core services are susceptible to the same effects of the relays, but although they are based on the unicast pipe, they behave differently. The bi-directional pipe has the same patterns and similar RTT to the unicast pipe. Other similarities are observed in earlier discussion of the direct communication. However, JXTA socket behaves more like the secure pipe, with comparable RTT over the relays.

6.5.2 Effects of Relays and Transports on Message Throughput

Figure 6.37 shows the achieved throughput through the relays for JXTA 2.2 setup on a WAN. The throughput values follow expected pattern due to the addition of relays and HTTP. The issues with the TCP transport module do not affect the one-way throughput. The most noticeable difference between JXTA versions is that the propagate pipe had a twice higher throughput via relay in JXTA 1.0, which dropped in JXTA 2.0.

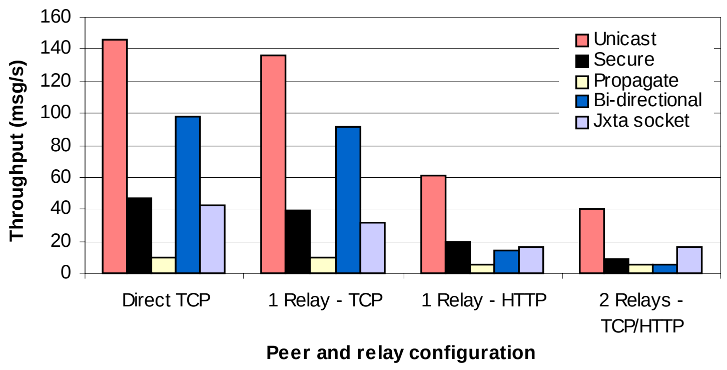


Figure 6.37: Message throughput through 2.2 relays

The propagate pipe also has the most stable throughput under different relay and network setups (close to 10 msg/s). The secure pipe improved in all performance aspects with JXTA 2.0 and that is reflected in relay throughput as well. While the JXTA socket still behaves similar to the secure pipe in most setups, the bi-directional pipe does not follow the earlier observed similarity to the unicast pipe. Its throughput is very much affected by the relays, noticeably more than unicast. It is also noticed that through the releases, the TCP relay improved over the HTTP in terms of sustainable message load.

The throughput increases with the increasing sending rate in a similar fashion for all pipe types, indicating a strong and consistent effect of the relay (Figure 6.38). The receiving rates are closely related to the relay's output rates and sustainable load. All types of pipes exhibit the optimal range, represented by the sharp cut-off of the curves in the graph. An attempt to send at a higher rate results in a lower receiving rate, usually accompanied by the message loss.

During the throughput tests, the JXTA 1.0 relay peers consistently failed due to overloading. The transmission would suddenly fail after about 3900 messages over unicast, 2800 over secure, and 2100 over propagate pipe. The repeated runs resulted in failures at almost the same message counts. The resource usage at the relay, in particular the creation of excessively high number of new HTTP connections to the receiver contributed to the problem. A similar problem was observed during the two-relay test runs on a WAN for both 2.x versions (secure, propagate and bi-directional pipes) and single HTTP relay for 2.2 (secure and propagate pipe), but only a few stable HTTP

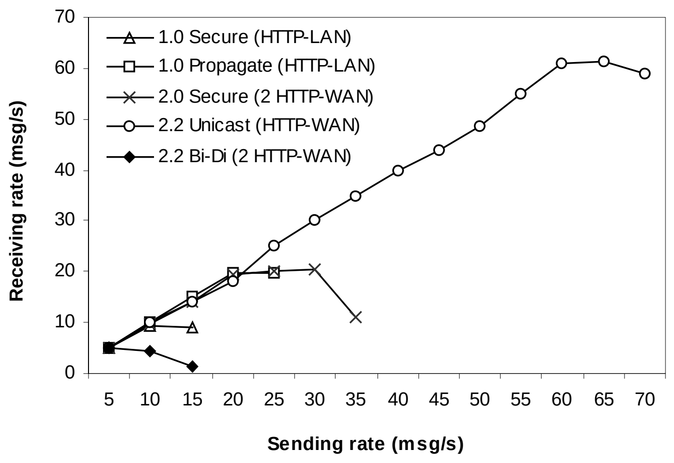


Figure 6.38: Relay throughput with overloading

connections are used throughout the test session. Several test runs of over 1,000 successful message transmissions were required to collect 10,000 samples.

6.6 Summary

This chapter reviewed the benchmarking results collected in a variety of peer configurations. The results for every component of the Performance Model expose the strengths and weaknesses of the JXTA platform. Several unintuitive results are observed, such as the impact of CM system on the centralized search performance and HTTP effect on typical peer operations and messaging through relays. Most of the results support the expectations and they can be used to more confidently assess the performance of the JXTA system. By collecting the presented performance results, the second goal of this thesis is accomplished.

Chapter 7

Lessons Learned

The JXTA Performance Model and Benchmark Suite are developed as the first goal of the thesis and used to collect numerous performance results about JXTA platform. The bottlenecks and limitations of JXTA components are presented throughout the analysis of the results completing the second goal. The third goal is achieved by looking at the big picture composed of all the individual collected results and is covered in this chapter.

7.1 General Observations

The performance results are collected in a variety of configurations, which are chosen as commonly used in networked and distributed systems, such as transport protocols, LAN and WAN environment. The ranges of the JXTA-specific parameters were chosen because they were expected to show the increasing cost or latency with higher workload, such as message sizes, peer group sizes and message complexity. Most of the parameter values produced logical and expected results and behaviors. However, a considerable number of unexpected and unintuitive results were obtained, which made this effort worthwhile. This shows that JXTA may be a still maturing P2P platform and that there are many opportunities to optimize a JXTA-based system. Finding the optimal configurations may not always be easy from the presented results, so it is recommended that the system designers use benchmarking in the early system analysis and design phases, with parameters that best suit their prospective system.

Apart from the performance results, this study used an implementation of a JXTA application [11], the benchmarks and also reviewed the evolution of the platform. The JXTA platform API seems to follow the standard Java conventions and models, but in early releases of version 1.0, there were numerous diversions from the standard models. Together with the general lack of documentation, it produced a steep learning curve for intermediate Java programmers. To date, it is still difficult to find an exact match between the current API, documentation and the tutorial for some components (e.g. JXTA socket). This situation has had some effect on this research and the actual benchmarking, adding a feeling of trying to hit the moving target at all times.

It is however encouraging to see that the later JXTA releases (2.0, 2.1, 2.1.1, 2.2, 2.2.1 and 2.3) move towards the stricter adherence to the Java programming conventions and models. This assumes changes in API, which were common since the early releases and required customization of the benchmarks to each tested JXTA version. Considering the open-source nature of the project, it is commendable that the compatibility between versions was preserved in all but one instance (upgrade from 1.0 to 2.0). Although JXTA 1.0 is already obsolete, the benchmarking results are collected for completion and comparison purposes, to better show the evolution of the platform. Regardless of the actual performance advantages in several cases, downgrading to JXTA 1.0 is not recommended, but the results can still be used to tune up the existing 1.0-based applications.

7.2 Deployment Patterns

The major lessons learned from the performance analysis are summarized in this section and presented using sample paths that may be used by a project manager to make decisions for a new or deployed system, and by JXTA developers to improve the platform. The first recommendation is that the general decision-making be prioritized by the effect that the component under consideration has on the whole system. The limitations of the rendezvous and relay peers are likely to have stronger consequences rather than pipe limitations, hence they should be considered first.

7.2.1 General Deployment Patterns

The decision path and discussion presented is offered as a general guide with no intention to recommend it for any particular type of system; it should rather be adjusted for specific needs.

This approach goes through the steps and components of the JXTA Performance Model. The starting point is the choice of the rendezvous peer(s) and the search model. The comparison of the search models clearly shows the superiority of the DHT-based approach vs. centralized or flooding. The evolution of JXTA towards the SRDI is logical and expected, considering that the DHT is a state-of-the-art solution for the large-scale systems [39, 45, 53, 61]. It should be noted that the centralized configuration does not occur naturally in JXTA 2.x. Since rendezvous peers only index the advertisements, the only way to have a rendezvous become a centralized search hub is to either have it pre-configured with advertisements or that an edge peer became a rendezvous after the peer group had lost the original rendezvous peer(s). Such situations may occur in a highly unstable environment or a strictly controlled peer group. Nevertheless, it is important to have the option of failing over to the efficient centralized model if required.

Although JXTA 1.0 is now obsolete, there are still deployed applications that were not upgraded to 2.x. To improve performance of such applications, several steps are suggested. For the centralized model, keeping the number of connected peers under 16 produces much better results. Splitting the peer network into smaller partitions, if possible, or even having the edge peers disconnect from the rendezvous for extended periods of inactivity can help. In case of thousands or more advertisements on a rendezvous, the problem of cache cleanup can be controlled by properly initializing the advertisement lifetime. It is important to avoid setting a too long lifetime that keeps the advertisement in the cache for much longer after the resource is no more available. Since this is a file-based cache, fast hardware should certainly help as well. To keep the query loss rates low, the only thing a programmer can do is reduce the amount of search requests, if possible. The deployment of smaller groups that generate lower query loads is another solution, which in addition reduces response time variation.

The flooding approach suffers from poor scalability with increasing query rate, group size and additional rendezvous peers. The query rate should not be allowed to exceed 4 q/s, and the number of rendezvous should be kept under four. Peer group sizes of up to 32 produce reasonable response times at low query rate (2 q/s), but for 4 q/s the group size of 16 already degrades performance.

In JXTA 2.x, the flooding approach is abandoned in favor of SRDI, but the centralized search is still possible. The major issue with centralized model is high memory consumption as the cache size approaches 1,000. The only proposed remedy is to use custom garbage collection scheduling for the JVM, but to apply it carefully so it does not degrade performance. For SRDI, the most interesting behavior is the positive effect of the second rendezvous, which seems to be the optimal improvement. Two rendezvous handle the high query loads extremely well, but adding more rendezvous is not so effective, it may even degrade the response time for lower query rates. The other important issue is the memory leak in the SRDI implementation, which currently requires restarts of the rendezvous peer. As far as the recovery after failure is concerned, there seems to be a definite advantage to using a stable seed rendezvous that the edge peers can rely on if all others fail.

The next step in the decision path looks at the relay peers. If the system or its component requires a relay, it is better to look at the constraints early to avoid the potential problems. For systems using JXTA 1.0 the best improvement would be to upgrade the system to the latest version. For JXTA 2.x, it is best to avoid HTTP relay configurations, when using secure, propagate or bi-directional pipes, due to the problems with pipe resolution and transmission reliability. Although the HTTP relay offers better message RTT, it is recommended to use TCP relay, if possible, due to better throughput and reliability, also because improvements are expected in the new version 2.3. Note that the choice of the transport protocol for the relay may affect the expected performance of the typical peer operations.

An additional consideration is that the lack of flow control will cause message drops at the overloaded relay, so the safest approach is to deploy a relay on the dedicated machine. The unicast and bi-directional pipes distinctively perform with highest throughput and stability over a TCP relay.

Deciding on the messaging implementation is the next step and can be made by looking at major performance advantages and disadvantages of each pipe. The unicast pipe is suitable for environments of high message loads and bursty traffic. Its overall performance scales well with the message size and number of senders, but its RTT performance degraded since the version 2.0 for small message sizes.

The secure pipe improved noticeably in version 2.0 and outperformed the unicast pipe in RTT of small message sizes (1 and 10 KB). It is very suitable for environments of two-way communication with low message load. On the other side, the secure pipe exhibits higher latency over WAN with larger message sizes of 100 KB and more.

The propagate pipe has a special feature in transparently allowing messaging from one sender to many receivers. This is an efficient way to propagate data, which is likely to be a primary reason for its selection. Although the performance of the actual RTT and throughput lags after other pipes, this may be only a secondary factor in the selection process. The results also show that the efficiency and value of the propagation increase with the addition of receivers, especially if the multicast is available. There is an additional benefit when rendezvous peers propagate messages. Receivers do not need to connect to the sender, whose only contact is with the rendezvous. This reduces the processing load on the sender, allowing it to run in a constrained environment, such as on a sensor that sends the readings to many interested receivers.

The bi-directional pipe is best for peer groups where two-way messaging communication is prevalent. Its API is also an advantage over the unicast pipe. However, it may be better to stick with unicast pipe in WAN environments of very high message load, where the bi-directional pipe suffers from high message loss.

A very similar in API, but with the low-level approach to data transfer, JXTA socket is a contender to the bi-directional pipe. It is a clear winner for two-way communication with message sizes up to 100 KB. The reliability adds to the advantages, but the major problem with JXTA socket is very low one-way throughput. However, although the sustainable load is lower than for unicast and bi-directional pipes, most of the systems would not put a single peer under the higher pressure that in can sustain.

To maximize performance of the typical peer operations, several options are available. The efficient use of the local advertisement cache brings the most benefit in saving the discovery cost. Careful tuning of the advertisement expiry time will make the cache maintenance faster and help avoid false representation of available resources. An extremely useful programming step is to check if the connection to the rendezvous is actually established before issuing the remote discovery command. Reusing open pipe handlers saves on resolution cost. This approach was used to optimize the messaging benchmarks, which shortened the test runs considerably.

7.2.2 Deploying for Secure Messaging

JXTA-based systems that require high security and in particular secure messaging would be designed and deployed to maximize performance of secure pipes. The benchmarking results show that secure pipes perform better in JXTA 2.x versions than in 1.0. They scale well with message size and number of senders. The RTT of secure pipes is excellent for message sizes of up to 10 KB, even on the WAN. For JXTA 1.0, complex messages with many elements should be avoided, but in JXTA 2.0 more small elements per message yields better RTT results. If the system requires the use of a relay, the best way to increase performance and reliability is to use a single relay with TCP for JXTA 2.2, HTTP for 2.0 and enable both TCP and HTTP for JXTA 1.0. The secure messaging choices need not affect the rendezvous setup, but the use of transport protocol may affect the performance of the typical peer operations.

7.2.3 Deploying for Messaging Reliability

The only tested messaging component with built-in reliability is the JXTA socket for JXTA 2.2. This messaging component is the best choice for systems that require the highest possible reliability. The configurations that maximize JXTA socket performance include messaging under 1 MB per element of transfer, and TCP relays for highest throughput and lowest RTT. It should be noted that the reliability of the JXTA socket comes at the price of lower throughput.

7.2.4 Suggested Platform Improvements

During the benchmarking, several issues came up that could not be circumvented by reconfiguring. Consultations with the members of JXTA community have traced these issues to the implementation problems and bugs in code [15].

All JXTA versions and search models suffer from the processing overhead of the incoming messages. The Project JXTA community has identified this issue and initiated a refactoring of the endpoint service for the next JXTA release (2.3), which is expected to resolve most of the problems.

The issues in thread synchronization of the incoming connections are responsible for the low query throughput of the rendezvous peers, although ample CPU power is available. It is recommended that these issues be dealt with very soon, due to the importance of the rendezvous peers and network.

To resolve the query loss problem, it is suggested that a flow control be added to the platform's rendezvous service implementation, which would pace the query senders and keep the query rate in check by the rendezvous peer. In addition, the recovery after rendezvous failure could be more aggressive to reduce waiting and periods of disconnection.

7.3 Summary

This chapter achieves the third goal of the thesis by demonstrating how the performance results can be combined to produce the deployment patterns for new applications and existing applications. Some changes in the configuration of the transports, rendezvous and relay peers can be made even on the deployed system, allowing the user or system administrator to optimize the performance of the JXTA system on the fly. Other changes and optimizations, especially for high security, must be made in the early implementation phases of the project to ensure that the system meets the requirements.

Chapter 8

Summary and Contributions

8.1 Summary

P2P systems introduce a new computing paradigm and emerge as a new type of distributed systems. They are characterized by the lack of centralized control, autonomous nodes, sharing of resources and intermittent connectivity. The most popular P2P applications are in the areas of worldwide file sharing and distributed computing. The available P2P applications are specifically designed to support a single type of service, such as file sharing and even within this category, the disjoint and incompatible networks are deployed.

Among different P2P systems and applications, JXTA has emerged as a novel technology with a goal to provide the standard protocols, their implementation and infrastructure for building all kinds of P2P applications. While it promises the abundance of features such as resource discovery, identification and group-based organization, its performance and scalability were not well understood. The goal of this research is to provide the model and tools for benchmarking JXTA components and to fill the gaps in the performance and scalability picture of the new platform.

JXTA is an infrastructure-rich P2P platform and its complexity allows many factors to impact its performance and scalability. This thesis investigates the performance issues of the JXTA platform through the proposed Performance Model and benchmarking of the JXTA reference implementation in Java. JXTA is evaluated through the typical peer operations, messaging features, search and discovery through rendezvous peers and message relaying. Benchmarking and evaluating JXTA is a rather complex endeavor. Since the performance of JXTA is strongly determined by the implementation, the most accurate and realistic analysis is possible through the actual real-life peer groups, rather than simulation. The benefits of this study are the actual performance and scalability characterization in smaller groups and the collection of realistic parameter values for use in a prospective JXTA simulator.

It is noted that the performance of the newer JXTA versions improved for some components and features and regressed for the other. Such results indicate that improvements in some components and addition of new features do not necessarily indicate the overall benefit of the newer versions and that all relevant features need to be evaluated before the major version upgrade on a deployed system. This study showed that there are numerous trade-offs between features and performance of JXTA components. Although this requires additional planning before implementing a system or application, it also offers ways to optimize performance of the existing or new systems.

8.2 Contributions

By conducting the performance study of the JXTA P2P platform, this thesis provides the following contributions:

- ❖ Introduction of the JXTA Performance Model provides the framework for performance evaluation of the existing and newly designed systems. The model covers the important components of JXTA and outlines relevant parameters for the in-depth analysis.
- ❖ Design and implementation of the Benchmark Suite provides the tools for performance evaluation of the JXTA components. The benchmarks take a variety of input parameters and allow the testing of JXTA components under different peer group configurations, messaging patterns and workloads, network environments, etc.
- ❖ Collected performance results reveal the bottlenecks and performance issues of JXTA from the early version 1.0 up until version 2.2, covering over three years of platform development. The results indicate the performance limitations and

characteristics of interest to the users, system designers, as well as JXTA platform developers. The observed results and conclusions are very well received by the JXTA community and the wider P2P research community where they were published [11-14].

- The results collected under various real-life configurations provide the bounds of JXTA component performance that can be applied as ranges of realistic and previously unknown parameters for the prospective JXTA network simulator.
- The deployment guidelines and paths are recommended as ways to maximize performance of JXTA systems, according to the specific application requirements.

8.3 Open Questions and Further Research

Although this study has put JXTA components through the in-depth testing and collected many results, it was not possible to test all combinations of configuration parameters. This section gives an outline of the major open questions that would make up the path of the further work.

The most important open question is about the scalability of JXTA in very large peer groups composed of thousands and even millions of peers. Apart from trying to deploy such a system with a large-scale application, the best approach for evaluating JXTA is a simulator. Such a simulator does not exist yet, but the results from this study can provide the ranges for the simulation parameters. The scalability of the rendezvous network in serving large peer groups is the primary question, but also the scalability of relays and pipes within the same environment.

The next points of interest are the behavior in sub-groups and the performance impact of HTTP in direct messaging and discovery. More detailed investigation of the rendezvous peer’s handling of different types of advertisements and wild-card searches is also recommended.

The JXTA community has initiated a number of projects to provide alternate language implementations, such as C and Perl. Evaluation of these implementations would be useful to see whether different implementation are more appropriate for some of the various devices and environments that JXTA aims at, such as small devices and sensors.

Within the area of P2P systems and platforms, it would also be interesting to investigate other P2P environments that offer features and functionalities similar to some of the JXTA components, and compare them to JXTA. Some products of interest are Windows Peer-to-Peer Networking [60] and Apple Rendezvous [41].

References

[1] Apache Xindice, The Apache Software Foundation, http://xml.apache.org/xindice/, 2003.

[2] O. Babaoglu, H. Meling, and A. Montresor, "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems," in Proc. The 22nd International Conference on Distributed Computing Systems, Vienna, Austria, 2002.

[3] S. Baehni, P. T. Eugster, and R. Guerraoui, "OS Support for P2P Programming: a Case for TPS," in Proc. The 22nd International Conference on Distributed Computing Systems, Vienna, Austria, 2002.

[4] BearShare, http://www.bearshare.com/, 2003.

[5] M. Castro, P. Druschel, A. M. Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, 2002.

[6] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility," in Proc. HotOS VIII, Schoss Elmau, Germany, 2001.

[7] EDonkey 2000, http://www.edonkey2000.com/, 2003.

[8] Gnutella.com, http://www.gnutella.com/, 2003.

[9] Groove Networks, http://www.groove.net/, 2003.

[10] P. K. Gummadi, S. Saroiu, and S. D. Gribble, "A measurement study of Napster and Gnutella as examples of peer-to-peer file sharing systems," *Computer Communication Review*, vol. 32, no. 1, 2002.

[11] E. Halepovic and R. Deters, "Building a P2P Forum System with JXTA," in Proc. The Second IEEE International Conference on Peer-to-Peer Computing, Linköping, Sweden, 2002.

[12] E. Halepovic and R. Deters, "The Costs of Using JXTA," in Proc. The Third IEEE International Conference on Peer-to-Peer Computing, Linköping, Sweden, 2003.

[13] E. Halepovic and R. Deters, "JXTA Performance Study," in Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, BC, Canada, 2003.

[14] E. Halepovic and R. Deters, "The JXTA Performance Model and Evaluation," *to appear in Future Generation Computer Systems*, 2004.

[15] E. Halepovic, R. Deters, and B. Traversat, "Performance Evaluation of JXTA Rendezvous," in Proc. Distributed Objects and Applications (to appear), 2004.

[16] S. Iyer, A. Rowstron, and P. Druschel, "SQUIRREL: A decentralized, peer-to-peer web cache," in Proc. 12th ACM Symposium on Principles of Distributed Computing, Monterey, CA, USA, 2002.

[17] Jabber Software Foundation, Jabber, Inc., http://www.jabber.org/, 2003.

[18] M. Junginger and Y. Lee, "The Multi-Ring Topology - High-Performance Group Communication in Peer-to-Peer Networks," in Proc. The Second IEEE International Conference on Peer-to-Peer Computing, Linköping, Sweden, 2002.

[19] M. B. Juric, T. Welzer, I. Rozman, M. Hericko, B. Brumen, T. Domajnko, and A. Zivkovic, "Performance assessment framework for distributed object architectures," *Advances in Databases and Information Systems*, vol. 1691, 1999.

[20] JXTA Bench Project, http://bench.jxta.org/, 2004.

[21] JXTA v2.0 Protocols Specification, http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html, 2003.

[22] K. Kant and R. Iyer, A Performance Model for Peer to Peer File Sharing Services, Intel Corporation, http://citeseer.nj.nec.com/kant01performance.html, 2001.

[23] B. Kantor and P. Lapsley, Network News Transfer Protocol, http://www.w3.org/Protocols/rfc977/rfc977.html, 2004.

[24] P. Karbhari, M. Ammar, A. Dhamdhere, H. Raj, G. RileyE, and l. Zegura, "Bootstrapping in Gnutella: A Preliminary Measurement Study," Georgia Institute of Technology GIT-CC-03-35, May 2003 2003.

[25] KaZaA, Sharman Networks, http://kazaa.com/, 2003.

[26] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An architecture for global-scale persistent storage," *ACM Sigplan Notices*, vol. 35, no. 11, 2000.

[27] N. Leibowitz, A. Bergman, R. Ben-Shaul, and A. Shavit, "Are File Swapping Networks Cacheable? Characterizing P2P Traffic," in Proc. 7th International Workshop on Web Content Caching and Distribution, Boulder, CO, USA, 2002.

[28] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer systems," in Proc. Twenty-First Annual Symposium on Principles of Distributed Computing, Monterey, CA, USA, 2002.

[29] LimeWire LLC, http://www.limewire.com/english/content/home.shtml.

[30] E. Markatos, "Tracing A Large-Scale Peer to Peer System: An Hour In The Life Of Gnutella," in Proc. 2nd IEEE/ACM International Symposium On Cluster Computing and the Grid, 2002.

[31] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," HP Laboratories Palo Alto HPL-2002-57, 2002.

[32] P. V. Mockapetris and K. J. Dunlap, "Development of the Domain Name System," *Computer Communication Review*, vol. 18, no. 4, 1988.

[33] MyJXTA2 Enterprise Edition, http://myjxta2.jxta.org/servlets/ProjectHome, 2003.

[34] W. Nejdl, B. Wolf, S. Staab, and J. Tane, "EDUTELLA: Searching and Annotating Resources Within an RDF-based P2P Network," in Proc. International Workshop on the Semantic Web, Hawaii, 2002.

[35] L. Olson, .NET P2P: Writing Peer-to-Peer Networked Apps with the Microsoft .NET Framework, http://msdn.microsoft.com/msdnmag/issues/01/02/netpeers/default.aspx, 2004.

[36] Project JXTA Community Home Page, http://www.jxta.org/, 2003.

[37] Project JXTA Solutions Catalog, http ://bench.jxta.org/ project/www/ Catalog/index-catalog.html, 2003.

[38] Project JXTA:Java™ Programmer 's Guide, Sun Microsystems, Inc., http://www.jxta.org/docs/jxtaprogguide_final.pdf, 2003.

[39] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Computer Communication Review*, vol. 31, no. 4, 2001.

[40] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-Aware Overlay Construction and Server Selection," in Proc. IEEE INFOCOM, New York, NY, USA, 2002.

[41] Rendezvous, Apple Computer, Inc., http://www.apple.com/macosx/features/rendezvous/, 2004.

[42] RFC-Gnutella 0.6, http://rfc-gnutella.sourceforge.net/developer/testing/index.html, 2003.

[43] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella Network," in *IEEE Internet Computing*, vol. 6, 2002.

[44] J. Ritter, Why Gnutella Can't Scale. No, Really., http://www.darkridge.com/~jpr5/doc/gnutella.html, 2002.

[45] A. Rowstron and P. Druschel, "File Systems - Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility," *Operating Systems Review*, vol. 35, no. 5, 2001.

[46] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 2001.

[47] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy, "An Analysis of Internet Content Delivery Systems," in Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI), Boston MA, USA, 2002.

[48] S. Saroiu, K. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in Proc. Multimedia Computing and Networking (MMCN), San Jose CA, USA, 2002.

[49] J.-M. Seigneur, G. Biegel, and C. Damsgaard, "P2P with JXTA-Java pipes," in Proc. The 2nd International Conference on the Principles and Practice of Programming in Java, Kilkenny City, Ireland, 2003.

[50] SETI@home, http://setiathome.ssl.berkeley.edu/.

[51] A. Sharma, The FastTrack Network, PC Quest, http://www.pcquest.com/content/p2p/102091205.asp.

[52] C. Shirky, What Is P2P... And What Isn't?, O'Reilly Network, http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html, 2003.

[53] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," *Computer Communication Review*, vol. 31, no. 4, 2001.

[54] P. Tran, J. Gosper, and A. Yu, JXTA and TIBCO Rendezvous – An Architectural and Performance Comparison, http://www.smartspaces.csiro.au/docs/PhongGosperYu2003.pdf, 2004.

[55] B. Traversat, M. Abdelaziz, M. Duigou, J.-C. Hugly, E. Pouyoul, and B. Yeager, Project JXTA Virtual Network, Sun Microsystems, Inc., http://www.jxta.org/project/www/docs/JXTAprotocols_01nov02.pdf, 2003.

[56] B. Traversat, M. Abdelaziz, and E. Pouyoul, Project JXTA: A Loosely-Consistent DHT Rendezvous Walker, Sun Microsystems, Inc., http://www.jxta.org/project/www/docs/jxta-dht.pdf, 2003.

[57] Universal Description, Discovery and Integration (UDDI), Oasis, http://www.uddi.org/, 2003.

[58] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov, Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment, Sun Microsystems, Inc., http://www.jxta.org/project/www/docs/mdejxta-paper.pdf.

[59] Web Services Description Language (WSDL) 1.1, WWW Consortium, http://www.w3.org/TR/wsdl, 2003.

[60] Windows Peer-to-Peer Networking, Microsoft Corporation, http://www.microsoft.com/windowsxp/p2p/default.mspx, 2004.

[61] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," University of California, Berkeley CA UCB/CSD-01-1141, April 2001 2001.

Appendix A Trademarks Used