The Costs of Using JXTA: Initial Benchmarking Results

Emir Halepovic
Department of Computer Science
University of Saskatchewan
emir.h@cs.usask.ca

ABSTRACT

Project JXTA is the first peer-to-peer application development infrastructure, which includes standard protocols and multi-language implementations. The performance and scalability of JXTA are not well understood, despite its widespread usage in the research community and increasing popularity in the industry.

This paper proposes a JXTA performance model and presents results obtained by benchmarking JXTA components. The results show the performance characteristics of the current JXTA reference implementation from a peer perspective. Important performance issues and trade-offs are identified and discussed, as a basis for the formulation of guidelines for system designers and simulation-based research of JXTA networks.

General Terms
Measurement, Performance.

Keywords
Peer-to-Peer, Distributed Computing, Performance, JXTA.

1. INTRODUCTION

Peer-to-peer (p2p) systems in general are characterized by decentralized control, high autonomy of participating nodes and heterogeneity in terms of processing power and shared hardware and software resources. Such systems are attractive for several reasons; among the most important are the lack of centralized control and a single point of failure, and the potential to scale to very large sizes.

Project JXTA [18] has joined the peer-to-peer family 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 to parallel computation [16, 27]. JXTA defines protocols for core p2p operations, such as discovery, messaging and group organization. Over two years of development in an open-source community have produced a solid API, largely tested in academic environment as a platform for prototyping various p2p concepts. Current statistics indicate that the JXTA community has grown to almost 13,000 members, with up to 20,000 downloads per week [18]. The existing applications of JXTA range from RDF-based resource discovery and retrieval [17], to p2p discussion forums [5]. The popularity of JXTA has spread into the market environment and some commercial products are already available [19]. However, the performance and scalability of JXTA are not well understood.

This paper will provide performance results of core JXTA components to both the research and developer community, as well as identify the components of a JXTA performance model. In a sense, this work represents a peer-centric view of the performance of JXTA. Specifically, message round-trip time (RTT), data throughput, and effects of various network and transport configurations are investigated. The results presented are useful for several purposes. First, the JXTA platform developers would benefit from establishing baseline performance results for the version 1.0 of the JXTA protocols implementation, as version 2.0 is being introduced. Second, the designers and users of p2p systems based on JXTA need to know the performance limits of their products, and need the guidelines to improve design and implementation. Finally, the p2p research based on simulation is provided with realistic parameters obtained from the real-life experiments.

This study is based on the latest and final stable release of JXTA 1.0 implementation for Java 2 Standard Edition (J2SE). The earliest stage of the work involved a design and implementation of the benchmark suite for evaluating JXTA components. The benchmarks are designed at the application level, so that they are reusable for future work on evaluating new JXTA versions.

The rest of the paper is organized as follows. Section 2 gives an overview of the JXTA platform. Section 3 describes the performance model of JXTA components and Section 4 presents the analysis of performance results. Related work is described in Section 5, and conclusions are given in Section 6.

2. JXTA OVERVIEW

Project JXTA is the first attempt to formulate core p2p protocols, on top of which interoperable p2p applications could be built [26]. Standardization of the common protocols would allow for easier interaction between heterogeneous peers. A lot of effort in the past years has gone into the implementation of application-specific protocols, such as for file sharing, instant messaging or collaboration. Standard protocols would provide the basic functionality for peer and resource discovery, communication and organization, which are necessary for all p2p applications. The details of JXTA protocols are provided in [12].

As a set of protocols, JXTA is independent of any programming language, operating system, device or underlying network transport [26]. JXTA is developed as an open-source effort, and uses open standards as its interoperability foundation, which is best represented by its dependability on XML. The reference implementation is available in Java language and C, Python, Perl and other language implementations are provided as community projects [18].

In general, p2p applications must handle intermittent connectivity, dynamic IP addresses and an unstable network topology. Therefore, the developers face a challenge of designing applications that are independent of DNS and IP addressing and identification. This is why JXTA introduces a virtual network

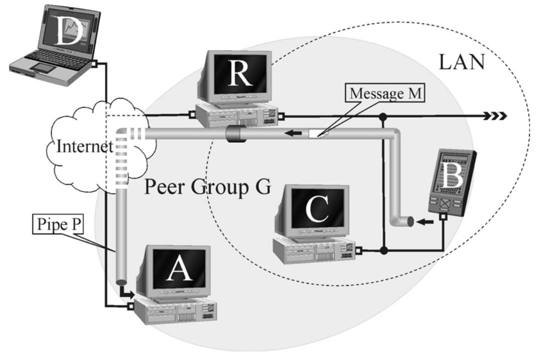

Figure 1: JXTA peer network

layer, on top of the existing transport protocols, with its own addressing and routing, facilitating interaction in this dynamic environment [26]. 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 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.

2.1 Peers and Peer Groups

The JXTA virtual network consists of several kinds of peers [20]. Most of the peers 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 the *proxy* peers (e.g. peer B), for caching, search and discovery. *Rendezvous* peers are usually more powerful peers, with well-known DNS name or stable IP address, and they act as caches of information about the connected peers. *Relay* peers have a special role to learn and provide routing information and pass messages between peers separated by firewalls and NAT. In Figure 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 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.

2.2 Advertisements

All entities in JXTA, including peers, groups, pipes and services, are represented by advertisements, which are XML documents of a well-defined format [12]. Advertisements carry a unique random ID number of the resource or entity they represent, and optional additional information, such as human-readable names and descriptions. 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 publisher 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 startup. This prevents a peer from attempting to access non-existent peers and services. Peer A from Figure 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. After the lifetime has expired, peer A should republish the service advertisement. If peer B joined the network after the advertisements 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 the advertisement processing and management impacts the performance of operations on the resources represented by advertisements.

2.3 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 bound to the physical location, IP address or a port. Instead, pipes have a unique ID, so each peer can carry its pipe with itself even when its physical network location changes. At runtime, a pipe end is resolved to an endpoint address to which it is currently bound.

In Figure 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 tells 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 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.

JXTA messages are XML-documents composed of well defined and ordered message elements [12]. The elements are name-value pairs, and they can carry any type of payload. 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 [12], composed of the 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 message exchange.

2.4 Rendezvous and Relay Peers

To support resource binding and the exchange of messages across networks and firewalls, two special concepts exist in JXTA: rendezvous and relay peers. Rendezvous peers agree to cache advertisements for their peer group, propagate messages and scope the advertisement query recipients. Rendezvous peers also provide a common location for peers from separate networks to exchange advertisements. They facilitate search and discovery and provide resolving operations, such as peer name resolution to an

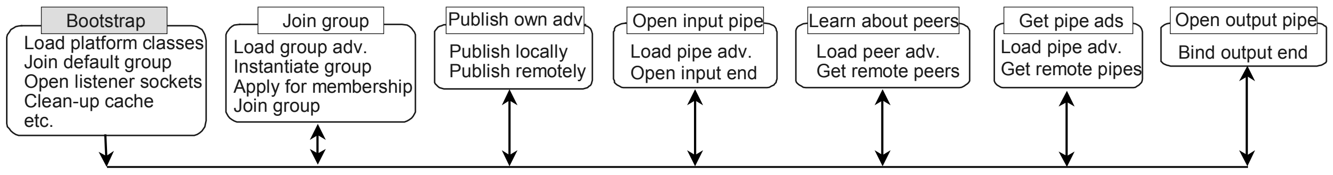

Figure 2: Typical peer operations

IP address [26]. Peer R acts as a rendezvous peer for group G in Figure 1. When peer A publishes the service or pipe advertisement, peer R propagates it to the other group members. When the peer B searches for the forecast service, peer R responds with matching advertisements or propagates the query to other peers, if no matching advertisement is found in its cache.

Relay peers, on the other hand, store routing information. Although JXTA messages contain routing information, relays are used when communication has to pass through a firewall. Relay peers can also spool messages for unreachable peers and serve as bridges between physical networks [26]. For example, since peers A and B are on different networks, they need relay R to pass messages between them. 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. PERFORMANCE MODEL

As a distributed and network-centric architecture, JXTA is evaluated primarily through its features of peer connectivity, communication and scalability. The connectivity and communication performance of JXTA are explored through a model consisting of the following components and metrics:

- Latency of typical peer operations
- Message round-trip time (RTT)
- Pipe message and data throughput
- Rendezvous peer query and response throughput
- Relay peer message throughput

These components and metrics are discussed and evaluated through their role in successful and effective deployment of a JXTA-based system. The benchmarks, metrics and discussion presented should also be applicable to any structured hierarchical p2p network, even the well-known popular file-sharing networks [4, 14].

3.1 Typical Peer Operations

A JXTA peer needs to perform a series of steps (operations) to join and participate in a JXTA network (Figure 2). Depending on the application and type of peer (edge, rendezvous or relay), the number and order of these steps may vary. The steps are dictated by the modular design of 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.

From the performance perspective, a developer would be interested in the cost of these operations due to at least two reasons, (1) high-cost of basic operations involved to complete a step, and (2) frequency of steps or operations performed. Note that most of the high-level steps are actually sets of distinct individual operations that a peer application may repeatedly perform during its lifetime, but the cost of a single step execution is measured for this study. In addition, the actual operations may be executed in different ways depending on the circumstances, but only the typical combination that a single step goes through is considered. The decision on what is considered typical has been strongly based on two applications: a p2p forum system [5] and MyJXTA [16].

The typical high-level steps and their basic operations are presented in the relative order a peer performs them (Figure 2):

1. ***Bootstrap the JXTA platform***.
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 our 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 interact with other peers.
7. ***Open output pipe*** to send messages to other peer.

1. *Bootstrapping the JXTA platform* loads a large class library, which may involve access to the local disk, network file system (NFS) or maybe even the Internet. This step builds the data structures to support the JXTA platform, cleans-up local cache and opens listener sockets, which are all time-consuming operations. Section 4.1 will show that bootstrapping is indeed very costly, compared to other steps, which could become an issue for peers with constrained resources.

During the bootstrap, all peers join the default hard-coded peer group, which is a universal worldwide peer group. Peers may choose to stay in this group, or join other groups. An edge peer also connects to the rendezvous and relay peers during bootstrap, if configured to do so. Once a peer bootstraps the platform, it may perform other steps, depending on the circumstances. Rendezvous and relay peers do not need to perform any other steps, because they are ready to serve the default peer group right after the bootstrapping. If they serve another user-defined group, then they proceed to join that group and open their services to the group members. For any peer, bootstrapping is the essential, first and complex step to perform, and it is therefore relevant for the performance study. A peer can bootstrap the JXTA platform only once inside the same Java Virtual Machine (JVM).

2. *Joining a group* is usually the next step after bootstrapping, and it is performed in JXTA by loading a peer group advertisement from 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, ranging from open access to LDAP authentication [1]. All communication with the rest of the peer network is done in the context of the peer group, even if that is just the default peer group. If a peer is joining the group for the first time ever, it must discover the peer group advertisement from the network or initiate a group by creating a brand 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 step for each group it wants to join.

3. A peer *publishes own advertisement(s)*, thereby announcing its availability to provide a service to other peers. Publishing has two types, local and remote. Local publishing puts the advertisement in the local cache, from where 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 to other peers and the rendezvous peer. This pushes the advertisement closer to other peers, which allows them a faster discovery. Depending on the frequency of publishing and number of connected peers, this may turn out to be costly in terms of network traffic and processing on the peers and rendezvous.

4. *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 disk. These two operations are treated as a single step because they are almost always performed together. A peer creates a new pipe advertisement only if the old one never existed or was destroyed. A good JXTA citizen would always reuse own advertisements, which other peers may have already discovered and cached. A peer would normally never retrieve its own advertisements from the network or its own cache.

5. The step of *learning about the peer group* is dependent 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 [5], a peer may be interested in just any other peer that is hosting a given forum. Although this step 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. A peer can search in own local cache for known peers (advertisements) or query the network for remote peers.

6. *Obtaining a pipe advertisement* of another peer varies in complexity and cost. In general, obtaining an 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, loaded from a user-specified file or URL etc. Frequently, a pipe advertisement is embedded within another advertisement published by a peer, such as a service or module advertisement. It is expected that a peer would use a local cache to save on a cost of discovering 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 the cache is not always an option. For example, JXTA advertisements have an associated lifetime, so unless a publisher refreshes an 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 in a certain JXTA deployment is to arrive at 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, and waste CPU cycles of requesters, responders and intermediate peers on the path.

7. *Opening the output pipe* means binding to the output end of a pipe for which another peer has opened the input end. No options are available for this step, it is required before sending a message and it may be costly. Binding a pipe end involves creation of connections between peers, several in case of relays, which imply latency.

The steps discussed are typical of group-structured or hierarchical p2p networks and systems, and also apply to the well-known file-sharing systems [4, 14]. Since JXTA is designed and implemented with structure in mind, especially group-based peer communities, it is desirable that peers follow this idea.

3.2 Message Round-Trip Time

Message round-trip time (RTT) is the most common metric used to evaluate a communication protocol or system. This paper investigates the application-level message RTT, which is affected by a number of important factors in JXTA. First, JXTA protocols are XML-based, which involves an overhead in the message composition and processing. Second, JXTA pipes present a high level of abstraction over underlying TCP and HTTP transports. Third, three different ways of message passing exist, unicast, secure unicast and propagate. Four, a path between peers may traverse one or more relays. Although the list of factors is not exhausted, these four are discussed here.

Measurements of RTT under different conditions and peer configurations indicate trade-offs between performance, security and reliability. Secure pipes are expected to introduce a delay penalty due to encryption, whereas propagate pipes by their nature introduce higher processing cost on both edge peers and relays. On the other hand, propagate pipes can exploit the availability of IP multicast to improve performance by avoiding the overheads of multiple connections associated with TCP and HTTP.

3.3 Pipe Throughput

As different layers of abstraction and overhead are introduced, the data throughput suffers. JXTA messages include a lot of overhead data, due to their XML-based structure and source-based routing. The question arises whether this overhead impairs data throughput significantly or not. Since JXTA messages do not have a size limit, it is expected that larger messages achieve higher data throughput. However, this depends on the message composition in terms of number of (XML) elements and the element payload size.

Table 1: Latency of typical peer operations (milliseconds)

Peer Configuration	Boot	Get group	Join group	Publish ads	Open in-pipe	Get local peers	Get remote peers	Get local pipes	Get remote pipes	Open out-pipe	Total
No Rdv	4465.8	1257.5	3.8	80.0	16.4	58.8	404.9	25.4	221.2	159.0	6692.7
No Rdv [with 40 ads]	8218.7	1337.2	3.9	81.8	20.7	61.8	392.9	28.4	236.1	173.5	10555.0
With Rdv on same LAN	5858.3	968.7	3.9	66.5	27.0	103.6	726.0	47.6	414.7	248.8	8465.3
With Rdv 6 hops away	6074.1	1190.1	4.3	65.8	16.5	123.3	1639.2	29.8	1173.4	693.2	11009.8

Table 2: Peer hardware and software configuration

Hardware	Software
PC AMD Athlon 800 MHz	MS Windows 2000 SP3
512 MB RAM	JVM 1.4.1 HotSpot
100 Mbps Fast Ethernet	JXTA Stable Release 092402

Ultimately, message throughput depends on the underlying TCP sending rate, but it is still important to know what the sending rate limits are in terms of number of messages. Moreover, both senders and receivers maintain message queues, which have their own effects. The queues amortize bursty traffic and affect reliability as messages may get dropped in case of overflow. JXTA pipes are unreliable and only higher-level services implement loss recovery.

3.4 Rendezvous Peer Throughput

Rendezvous peers serve several purposes and can potentially be subject to high message load. Primarily, rendezvous peers respond to search queries from other peers. In addition, they propagate messages within the peer group and take part in pipe resolution. The response time, message and query throughput and advertisement cache management are all important performance factors for a rendezvous peer. The effects of cache on the rendezvous bootstrapping are briefly explored, and the full rendezvous benchmarking is left for future work.

3.5 Relay Peer Throughput

Relay peers' main purpose is to pass messages between peers that cannot establish direct connection. As the number of firewalls and NATs increases in the Internet, the connectivity becomes more of a problem, and the role of relays becomes more important. Relays pass messages and cache routing information, so they are potentially exposed to large amounts of traffic. Therefore, scalability and performance of relays dictates the size of the peer group they can support and the message load their group can safely sustain. Both message load and the type of pipes used for message transfer affect the relay. In large peer networks, the question may not be how powerful machine a relay should be, but how many relays are necessary for making the system work.

4. RESULTS AND ANALYSIS

In this section are presented and analyzed the results obtained by measuring the components of the proposed performance model (Table 1). The measurements are obtained by running a series of benchmarks. The benchmarks are designed to run at the application level; hence they measure the application's "perspective" on JXTA performance. The benchmarks are reusable for future versions of JXTA, given the stability of the small subset of the Java implementation API. The hardware and software running on the peers in all tests is listed in Table 2.

4.1 Peer Operations

A peer typically performs a series of steps during the course of its lifetime in a peer network. The measurements of the time required to accomplish these steps were taken with the goal of identifying the cost of each step relative to other steps and the startup process. Although the order and frequency of the steps is application-specific, it is still worthwhile looking at some typical scenarios and combinations of steps.

In a hypothetical scenario, an edge peer performs the given steps once upon startup in the order shown in Figure 2. Some steps are split into basic operations to show their relative difference in latency. When all of the given steps are completed, a peer has joined the group, learned about member peers and resources, and decided with which peer it wants to interact. This scenario is in part based on the p2p forum application from [5], and in part on MyJXTA application [16].

Four peer configurations are compared, two for peers in ad-hoc groups with different cache sizes, and two for peers using rendezvous/relay at different distances, both without any extra advertisements in the cache (Table 1). Two edge peers are used for the ad-hoc tests, one is taking measurements and the other provides discovery responses. A rendezvous/relay peer is added for the additional tests; one on the same LAN, and the other 6 hops away. The remote rendezvous has a high-speed cable modem connection to the Internet, with the average round-trip time measured by *ping* tool of 67 ms, compared to less than 1 ms for LAN traffic. Presented results are based on 10,000 measurements of the given sequence of operations.

Bootstrapping: Assuming that a starting sequence includes a single execution of each step, bootstrapping itself consumes most of the total time. This step is mostly affected by the cache size and the use of a rendezvous/relay. Connecting to the rendezvous/relay adds the overhead of several socket connections. Depending on the distance, the measurements show that connecting to the rendezvous may extend the boot process by as much as 36%. However, the cache size has much stronger effect on the bootstrapping performance, and it is explored in more detail.

Local advertisement cache: Although local caching is not mandated by the JXTA protocol specification [12], its benefits are obvious when comparing the time required for retrieving local vs. remote peer and pipe advertisements. The trade-off comes primarily with the cost of bootstrapping. To exploit the cached information, a peer needs to keep the cache up to date. The cache is cleaned up during the bootstrapping, which comes at high cost. It takes almost twice as long to start with 40 advertisements in the cache, compared to starting with the empty cache. For

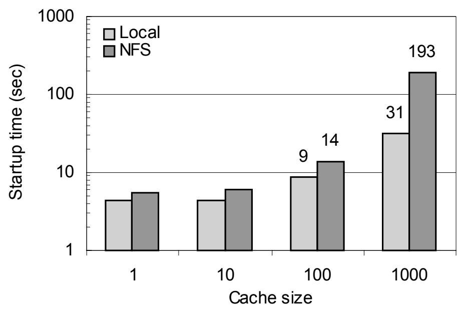

Figure 3: Peer startup time for different cache sizes

applications where peers restart frequently, the overhead of cache maintenance may adversely affect user's experience.

In addition, the "freshness" of the cache is indirectly affected by the behavior of other peers, in terms of correctly specifying lifetime of the advertisements. Peers should specify the lifetime so that it matches both their own lifetime in the group and their intention to periodically refresh the advertisements.

The effect of the cache size is best seen through the time it takes a rendezvous/relay peer to bootstrap the platform with different cache sizes (Figure 3). The bootstrap time is measured for the cache located NFS and on local disk. Note that the growth of values for NFS is at much higher rate than for a local disk. For a rendezvous peer starting with 1000 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.

Joining a group: The process of joining was measured separately for creation of a group and the actual join operation by invoking the required methods. The join in this test uses an empty credential, which essentially means that a peer is joining an open group. That is why the join is so fast compared to the group instance creation. A peer may join using an LDAP authentication or some other mechanism, which would likely take more time, but this test is performed around the minimal required operations. Note that the group creation involves retrieval of the group advertisement from cache and creation of the group object, which sets up the environment for the group inside the JVM. This is the second most expensive operation overall and that is an issue of 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 peers and pipes (or any other resources) is an order of magnitude slower than retrieval from local cache (Table 1). Note that the results in this test present the time until the first discovery event, which certainly does not mean that this event would give a full picture of the discovered resources in the peer group. 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. The question arises what is to be done in case a peer cannot rely on the cache to provide the accurate information. In this case, a rendezvous peer should be able to provide an accurate picture of the peer group, at least in terms of the 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 connections rely on 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 messaging cost (Table 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 certainly worth it if the application requires that peers have updated information about the group.

Opening output pipes: To open a pipe, its name must be resolved to an endpoint, possibly using a rendezvous and a relay, and it is strongly dependent on the network distance between peers. Combined with the remote advertisement discovery, it turns out that connecting two peers on a LAN may take in excess of 600 ms; across the Internet several seconds. Developers should therefore try to reuse both the advertisements and open pipe handlers, whenever possible. Savings by reuse are higher if peers cannot connect directly and must use a relay as an intermediary.

Publishing advertisements: The results from Table 1 indicate that the last step in this discussion is certainly not the least important. The apparent low cost of this operation does not mean there is nothing to discuss, because there is no significant cost. On the contrary, this step 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 one of the foundations of the concept of peer-to-peer computing. Published advertisements are sent to the rendezvous and, depending on the network and transport configuration, to other peers in the group as well. If a rendezvous does not have the desired advertisements, it must contact all other peers before it can respond. So by publishing, a peer both raises the awareness of other peers about the available resources and saves them on cost of discovery. When used properly with tuned advertisement lifetime settings, performing this step goes a long way towards a more efficient peer network.

4.2 Message RTT

The measurements of the message RTT were obtained in a setup of two edge peers and if required, one relay, all on the same LAN. The RTT represents the time elapsed between the send request and the receipt of the acknowledgment at the sender. The messages are sent sequentially, and each message includes the sender's pipe advertisement and the message sequence number for the total of 316 bytes. It is actually very common that a JXTA peer sends own pipe advertisement inside the message, so that a receiver may respond. Processing is minimized at the receiver by reusing the open response pipe. The tests are performed in various configurations to show the effects of type of pipe, message size and composition, transport protocols and relays. All results are based on 10,000 acknowledged messages with 1,000-message warm-up period.

4.2.1 RTT and Message Size

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 of size ranging from 1 KB to 10 MB. The messages were exchanged between two peers using a

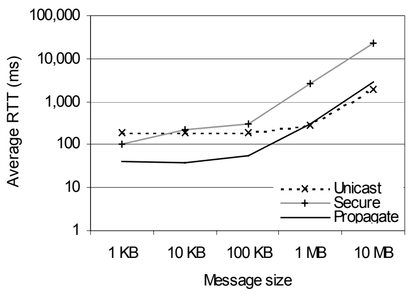

Figure 4: Average message RTT for various message sizes

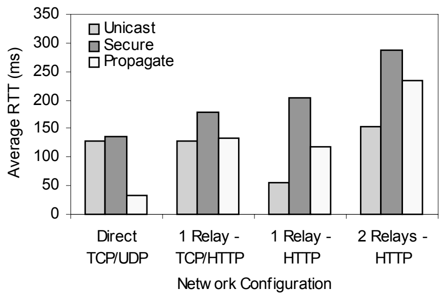

Figure 5: Average message RTT for various transports and relay configurations

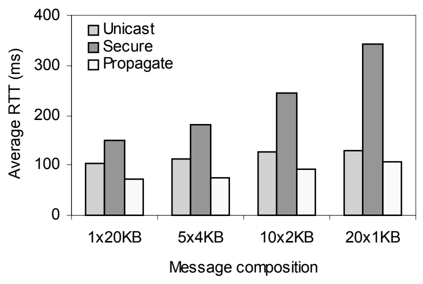

Figure 6: Average message RTT for various message compositions

direct TCP connection. Figure 4 shows the average RTT of three types of JXTA pipes for each of the message sizes (both axes are log scale).

Each pipe exhibits slow, close to linear increase in RTT from 1 KB to 100 KB, unicast pipe even to 1 MB. 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 follow a linear trend. The increasing rate for unicast pipe from 1 MB to 10 MB closely follows the rate of other pipes, but it is hard to conclude what would be its possible ongoing trend. It is certainly desirable that message RTT over any pipe increases at most linearly over message sizes. The graph of the maximum RTT (not shown) looks very similar to the graph of the average RTT.

By relative comparison, the propagate and secure pipes differ in the order of magnitude consistently, whereas unicast pipe performs closer to secure for smaller messages, but converges with propagate pipe for larger sizes. This indicates better scaling properties of the unicast pipe over TCP up to the 10 MB message size.

The results for the larger message sizes are not available because it was not possible to transfer messages of 100 MB and larger in our setup, regardless of the abundance of memory or tuned JVM setting. The maximum message sizes successfully transferred at least once were 86 MB over unicast, 97 MB over secure and 29 MB over propagate pipe.

4.2.2 Effects of Relays and Transports on RTT

JXTA peers can be configured to use TCP, HTTP and UDP in any combination. TCP is used whenever a direct connection between peers can be established. HTTP is used when a firewall or NAT is on the path or a peer specifically wants to use a relay, and UDP is exploited in the form of IP multicast for efficient propagation of messages on a subnet. The measurements of the RTT in different peer configurations and on different transports are shown in Figure 5. The messages in this test contain one payload element 1 KB in size.

In the Direct TCP/UDP configuration, two peers connect directly without any rendezvous or relay. Propagate pipes show much better performance because the peers use available IP multicast.

In the second configuration, two peers use a relay on the same LAN, all configured to use both TCP and HTTP, with multicast disabled. The RTT for unicast pipe, which is almost the same as for the direct TCP connection, indicates no significant effect of the relay. The secure and propagate pipes show higher latency when using a relay, as expected. While the increase in latency over a secure pipe is moderate, it quadrupled for propagate pipes. This large increase for propagate pipe is affected mostly by removing multicast.

The third configuration reveals more interesting results. Here the peers use only HTTP to communicate with the relay. Unicast pipes perform much better here than in other configurations, which is quite unexpected, considering overhead of HTTP and a relay. Similar result was obtained for propagate pipe, but the difference is minor compared to the second configuration. Secure pipe is slower as expected, again by a moderate amount.

Finally, the two-relay configuration is set up so that relays have both TCP and HTTP enabled, and edge peers use only HTTP to connect to one relay each. The relays pass messages between each other on behalf of their attached edge peers. Secure pipe gets uniformly slower through the configurations as expected. Compared to the configuration with HTTP and one relay, unicast and propagate pipes also show higher latency, as expected. However, when compared over all four configurations, unicast pipe performs almost equally throughout except in a single relay setup with only HTTP transport. Propagate pipe, on the other hand, suffers most from the additional relay, passing messages twice as slow as with a single relay. Overall, a single HTTP relay costs 4 times, and two relays 8 times more than IP multicast for propagate pipe.

4.2.3 RTT and Message Composition

The following test looks for difference in performance of pipes depending on the number and size of the message elements they carry. Four variations of a 20KB message were used: 1x20KB (1 element of size 20KB), 5x4KB, 10x2KB and 20x1KB.

Figure 6 shows that the performance of secure pipes is very much affected by message composition. On the other hand, there is less effect on unicast and propagate pipes, but it does exist. Increasing the number of elements from 1 to 20, while reducing their size, more than doubles the average RTT on a secure pipe. For unicast pipe, the increase in RTT over four combinations is 24%, and for propagate pipe about 43%. This effect of message composition is not very surprising, considering that messages are XML documents and the complexity of document structure affects the processing time. In addition, using more message elements to transfer the same amount of payload (in bytes) increases the overall message size by additional XML element tags, but this is not a very significant factor for smaller messages. In case of secure pipes, the encryption cost rises rapidly with increasing

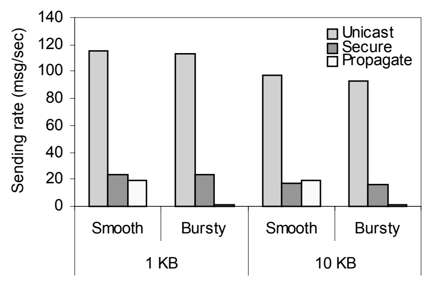

Figure 7: Pipe maximum sending rate

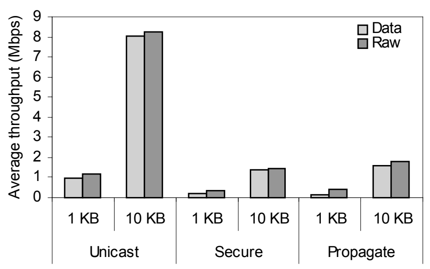

Figure 8: Average pipe throughput between two peers

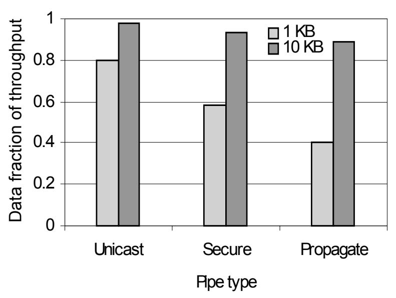

Figure 9: Payload data as a fraction of total throughput

amount of data. Therefore, the composition of messages is yet another variable to consider when designing a JXTA-based system.

4.3 Pipe Message and Data Throughput

JXTA pipes are by definition unreliable, and this lack of reliability is largely reflected by the queuing policy. All messages are queued before actual sending at the sender's side, as well as on the receiver's side before delivering to the application layer. If the queue is full, the oldest message is discarded to make room for the new one. Since JXTA pipes use TCP and HTTP, they tend to be reliable, only the propagate pipe lacks reliability with IP multicast, but it behaves same as the other two types when multicast is not available.

4.3.1 Pipe Message Throughput

The test of pipe throughput measures the limits of reliable transfer in JXTA terms, meaning that no messages are dropped. Both sender and receiver rates are included. The most interesting feature exhibited by unicast and secure pipes is the limited sending rate, regardless of the attempted rate. The sending rates of up to 300 messages per second (msg/sec) were attempted, but they were capped at levels shown in Figure 7. For unicast pipes, the maximum sending rate measured at 115 msg/sec for 1 KB, and 97 msg/sec for 10 KB messages delivered at smooth rate by the application. Under the same conditions, a secure pipe sends at about 23 msg/sec 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 consisted of 50-message bursts and 500 ms sleep time. No significant difference was observed between the rates for 1 and 10 KB message (under 4.6%), meaning that it is the number of message, not the number of bytes that affects the sending rate.

For propagate pipes, the sending rate of about 20 msg/sec is both the maximum no-loss rate and the rate that an application should attempt. Higher sending rate causes messages to be dropped from the sender's queue. The almost non-existent value for bursty traffic a propagate pipe can sustain indicates the failure to accommodate any significant burstiness. At most 2-message bursts followed by an idle period resulted in maximum rate without drops at the sender, but still had a small drop rate at the receiver.

The obtained results for limits on sending rate are useful for both the simulation of JXTA networks and the design of JXTA applications. For example, for a given number of peers in a group, it is possible to calculate the maximum message load peers can produce. In the simulation, given the network load, it is possible to calculate the number of peers required to generate the wanted traffic.

On the receiver side, no message drops were recorded for maximum sending rates over unicast and secure pipes. For propagate pipe, significant drop rates were recorded, depending on the sending rate, message size and burstiness. An attempt to send at the rate of 200 msg/sec over a propagate pipe translated into a 42.9% message loss over a session of 10,000 messages. When sending at the rate of 20 msg/sec, the loss was ranging from zero to 23.95% over 10,000 messages for different message sizes and burstiness, in an inconsistent fashion. The only conclusive behavior from the measurements is that the propagate pipe always performed without loss at smooth sending rate of 20 msg/sec.

4.3.2 Pipe Data Throughput

Considering a relatively low no-loss rate in msg/sec, it is important to look at its impact on the data throughput of JXTA pipes. As Figure 8 shows, it is not surprising that larger messages achieve higher throughput, because the number of messages is the limiting factor, not the size. At the same time the difference between the raw vs. data throughput is recorded. Data throughput measures the actual payload transmitted in the message, and it is considered an important metric in the performance evaluation of distributed object architectures [10].

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. In Figure 9 are shown the fractions of the throughput that are actual payload data. 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.

4.4 Relay Message Throughput

Relays are necessary to connect the pipe ends between peers and they introduce the overhead of communication by adding the processing cost and extending the pipe length. The relay throughput test measures the receiving rate of messages, given some sending rate and the message path through relay. The expected lower receiving rates can quantify the overhead a relay imposes on pipe throughput.

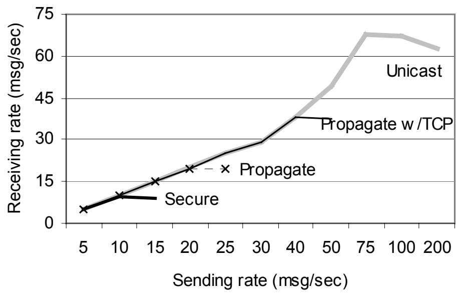

Figure 10: Maximum receiving rate through relay peer

The setup of this test consists of two HTTP edge peers connected to the relay, all on a LAN. The messages are sent from one peer to the other over all types of pipes. The message throughput is measured for no-loss transmissions and shown in Figure 10 (note that x-axis is not linear scale).

The throughput increases at a very similar rate for all pipe types, indicating the strong and consistent effect of the relay. The receiving rates are closely related to the relay's output rates and they are lower than for the direct peer connection. The maximum receiving rate through the relay was measured at 67 msg/sec for unicast, and just over 9 msg/sec for secure pipe. These results represent up to 41% and 60% reduction in message throughput for unicast and secure pipe over the direct TCP connection, respectively. The smallest effect is seen on propagate pipes, which still keep their throughput at about 20 msg/sec. However, if the sender is configured to use both TCP and HTTP, the rate almost doubles for propagate pipes, but with a large number of out-of-order messages at the receiver.

All types of pipes exhibit the optimal transmission rate, represented by the sharp cut-off in the graph. An attempt to send at a higher rate results in a slightly lower receiving rate, likely due to overloading of the relay.

Only throughout this test, the relay consistently failed, apparently 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 would result in failures at almost the same message counts. The speculated reason for this would be the problems in resource usage at the relay, in particular the creation of excessively high number of new HTTP connections to the receiver, possibly for each transmitted message.

5. RELATED WORK

The peer-to-peer approach has gained significant success and popularity in recent years, mostly due to several widespread types of application: instant messaging (IM), file sharing and collaboration. Most of the p2p applications have a very specific way of dealing with peer and resource discovery, communication and resource sharing. On the other hand, JXTA primarily provides standard protocols and tools to build interoperable p2p application regardless of their type. A related effort to standardize the protocols for one type of application is Jabber [8]. Jabber defines the open-standard protocol for message exchange, offering extensible and decentralized IM solutions. Gnutella and FastTrack [14] offer their own protocols for worldwide file sharing, using hierarchical network with super-peers to facilitate search and discovery. In the lower-level distributed object world, CORBA represents a solution for heterogeneous object registration and discovery, and remote method invocation.

Solutions for efficient large-scale object location and routing are provided in the form of application-independent protocols, such as CAN [21], Pastry [3] and Tapestry [15]. These protocols are based on distributed hash tables (DHT) and they can be used to build application-level multicast [3], distributed file storage [15] and cooperative web caching [7]. The DHT protocols fit on top of JXTA as higher-level services. They could provide a more efficient search mechanism, while exploiting the lower-level resource discovery features of JXTA [26]. Business-oriented systems, such as Web Services, are based on open standards of XML, SOAP and WSDL [6], in which centralized UDDI directories facilitate the search, but they do not support dynamic discovery and concept of group organization.

Some of the existing p2p solutions are not built with high performance and scalability in mind [21, 22]. This has prompted research into the performance, scalability and characterization of p2p systems. Such work includes analytic and simulation-based performance modeling [13] and measurement studies based on network crawling and traffic tracing [22, 24] of Gnutella network. The traffic characterization study was based on a campus-level trace of several content delivery networks [23]. The available 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 environment.

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, the JXTA-wire (many-to-many pipe) was evaluated for support of Type-based Publish-Subscribe approach for building p2p applications [2]. The JXTA propagate pipe was compared to the alternative solution for high-speed communication within peer groups [9]. In the previous work, the peer discovery and unicast pipe performance in the context of a p2p forum system were investigated [5].

Earlier results indicate that a broad and more detailed performance study is needed. The JXTA community initiated a dedicated Bench sub-project, with a purpose to collect performance and scalability measurements as the platform development progresses [11]. The results of the various measurements are published on the project web site in the form of time-series graphs and progress summary. The majority of the information is provided in absolute numbers, most appropriate for the platform developers. The emphasis is currently put on the measurements of the pipe throughput and rendezvous search and discovery performance, showing progress over JXTA releases. The test results are from the controlled environment. The Bench project provides the evaluation JXTA pipes in more detail, considering differences in type of pipes, earlier JXTA versions and the operating system [25].

The work and results presented in this paper are intended to complement the existing efforts, and extend them in terms of more detailed examination of JXTA components. The development of the benchmark suite and collection of results provides indication of the areas that need improvement in JXTA and guidelines for system designers to build better p2p applications. Although the discussed measurement methods and tools are presented in the context of JXTA, they are also applicable to any current or future structured hierarchical p2p systems.

6. CONCLUSION

This paper discussed the performance issues of the JXTA platform and presents the performance model and the benchmarking results for the reference implementation. The results are obtained from JXTA peer configurations on a single LAN, in almost all tests. The absolute measurements obtained are therefore most applicable to the enterprise deployment of JXTA. Nevertheless, it is expected that the general characteristics of JXTA components observed and their relative comparisons would translate to the wide-area deployment.

This study investigates typical peer operations, local cache and its underlying file storage, pipe and advertisement reuse, and various messaging parameters. The obtained results show high cost of the JXTA bootstrapping process relative to other operations, with the major factor being local cache management. In particular, the significant negative impact of NFS is noted. Local cache, on the other hand, allows for significantly faster gathering of information about the peer group resources, especially when combined with the disciplined advertisement publishing. The high cost of pipe binding strongly suggests the reuse to developers.

The relative performance cost of different types of pipes is analyzed in respect to message size and composition, network transports and relays. The most important observations are the good scaling properties of message RTT over various message sizes, sending rate limits and the limits of reliable message throughput. The significant impact of relays on message RTT throughput and reliability is also measured.

With the presented results, it is possible to derive some guidelines for the developers of distributed applications based on JXTA. For example, relatively low message throughput combined with the effects of message composition and its structure overhead suggests that it is more efficient and safer to transmit moderate-sized messages and minimize the traffic. File-sharing applications could benefit by carefully choosing the most efficient transfer fragment size. It further seems important to reduce the number of relays on the message path as much as possible. This result indicates that peers from separate networks should pick one relay from one of the networks, rather than having a separate relay in each network. Deeper analysis and derivation of recommendation and guidelines should provide more valuable results, and it is left for future work.

By performing the evaluation of JXTA components on a LAN, only a part of needed testing and benchmarking was tackled. Areas of future work include primarily scalability evaluation of large peer groups, in terms of messaging through relays and discovery on rendezvous peers. For JXTA pipes, the throughput with multiple senders for unicast and secure, and multiple receivers for propagate pipe should be measured and compared to one-to-one communication. In addition, the testing of all components on a wide-area configuration would be necessary to complete the performance and scalability picture of JXTA.

7. REFERENCES

[1] Altman, J. PKI Security for JXTA Overlay Networks, IAM Consulting, Inc., http://www.jxta.org/docs/pki-security-for-jxta.pdf.

[2] Baehni, S. *et al.* OS Support for P2P Programming: a Case for TPS. in *ICDCS 2002* (Vienna, Austria, 2002).

[3] Castro, M. *et al.* Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, *20* (8). 1489-1499.

[4] Gnutella.com, http://www.gnutella.com/.

[5] Halepovic, E. *et al.* Building a P2P Forum System with JXTA. in *P2P '02* (Linköping, Sweden, 2002).

[6] Hoschek, W. The Web Service Discovery Architecture. in *IEEE/ACM SC 2002* (Baltimore, USA, 2002).

[7] Iyer, S. *et al.* SQUIRREL: A decentralized, peer-to-peer web cache. in *PODC 2002* (Monterey, CA, USA, 2002).

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

[9] Junginger, M. *et al.* The Multi-Ring Topology - High-Performance Group Communication in Peer-to-Peer Networks. in *P2P '02* (Linköping, Sweden, 2002).

[10] Juric, M.B. *et al.* Performance assessment framework for distributed object architectures. *Advances in Databases and Information Systems*, *1691*. 349-366.

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

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

[13] Kant, K. *et al.* A Performance Model for Peer to Peer File Sharing Services, Intel Corporation, http://citeseer.nj.nec.com/kant01performance.html.

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

[15] Kubiatowicz, J. *et al.* OceanStore: An architecture for global-scale persistent storage. *ACM Sigplan Notices*, *35* (11). 190-201.

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

[17] Nejdl, W. *et al.* EDUTELLA: Searching and Annotating Resources Within an RDF-based P2P Network. in *Semantic Web Workshop* (Hawaii, 2002).

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

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

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

[21] Ratnasamy, S. *et al.* A scalable Content-Addressable Network. *Computer Communication Review*, *31* (4). 161-172.

[22] Ripeanu, M. *et al.* Mapping the Gnutella Network. *IEEE Internet Computing*, 6 (1) 2002, 50-57.

[23] Saroiu, S. *et al.* An Analysis of Internet Content Delivery Systems. in *OSDI '02* (Boston MA, USA, 2002).

[24] Saroiu, S. *et al.* A Measurement Study of Peer-to-Peer File Sharing Systems. in *MMCN '02* (San Jose CA, USA, 2002).

[25] Seigneur, J.-M. Jxta Pipes Performance, http://bench.jxta.org/papers/jmjxtapipesperformance.pdf.

[26] Traversat, B. *et al.* Project JXTA Virtual Network, Sun Microsystems, Inc., http: //www.jxta.org/project/ www/docs/JXTAprotocols_01nov02.pdf.

[27] Verbeke, J. *et al.* 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.