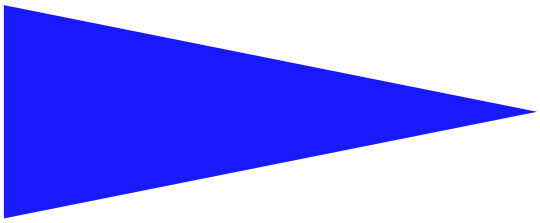
A practical example of convergence of P2P and grid computing: an evaluation of JXTA's communication performance on grid networking infrastructures

Gabriel Antoniu, Mathieu Jan, David Noblet

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1749



A PRACTICAL EXAMPLE OF CONVERGENCE OF P2P AND GRID COMPUTING: AN EVALUATION OF JXTA'S COMMUNICATION PERFORMANCE ON GRID NETWORKING INFRASTRUCTURES

GABRIEL ANTONIU, MATHIEU JAN, DAVID A. NOBLET

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

A practical example of convergence of P2P and grid computing: an evaluation of JXTA's communication performance on grid networking infrastructures

Gabriel Antoniu, Mathieu Jan, David A. Noblet

Systèmes numériques
Projet Paris

Publication interne n1749 — September 2005 — 33 pages

Abstract: As the size of today's grid computing platforms increases, the need for self-organization and dynamic reconfiguration becomes more and more important. In this context, the convergence of grid computing and peer-to-peer (P2P) computing seems natural. However, grid infrastructures are generally available as a hierarchical federation of LAN-based or SAN-based clusters interconnected by high-bandwidth WANs. In contrast, P2P systems usually run on the Internet, on top of random, generally flat network topologies. This difference may lead to the legitimate question of how adequate are the P2P communication mechanisms on hierarchical grid infrastructures. Answering this question is important, since it is essential to efficiently exploit the particular features of grid networking topologies in order to meet the constraints of scientific applications. This paper evaluates the communication performance of the JXTA P2P platform over LANs and high-performance SANs and WANs, for both J2SE and C bindings. We discuss these results, then we propose and evaluate several techniques able to improve the JXTA's performance on such grid networking infrastructures.

Key-words: Peer-to-peer, grid computing, JXTA, communication layers, performance.

(Résumé : tsvp)

This paper has been submitted to a special issue on Global and Peer-to-Peer Computing of Journal of Grid Computing.

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Un exemple pratique de convergence entre P2P et calcul sur grille: évaluation des performances des communications de JXTA sur une infrastructure réseau de type grille

Résumé : Étant donnée la taille de plus en plus grande des grilles de calcul d'aujourd'hui, le besoin d'auto-organisation et de reconfiguration dynamique devient de plus en plus important. Dans ce contexte, la convergence du calcul sur grille et du calcul pair-à-pair (P2P) semble naturelle. Toutefois, une infrastructure de type grille est généralement constituée d'une fédération hiérachique de grappes connectées par des réseaux LAN et SAN et interconnectées par des réseaux WAN à haut-débit. En comparaison, les systèmes P2P sont généralement déployés sur Internet, une topologie réseau habituellement plate. Cette différence peut amener à s'interroger sur l'adéquation des mécanismes de communication P2P sur de telles grilles de calcul hiérarchiques. Répondre à cette question est important, puisqu'il est essentiel d'exploiter efficacement cette particularité topologique des grilles de calcul afin de satisfaire les contraintes des applications scientifiques. Ce papier évalue les performances des communications des implémentations J2SE et C de la plate-forme logicielle pair-à-pair JXTA sur des réseaux de type LAN et des réseaux haute-performance SAN et WAN. Nous discutons ces résultats, puis nous proposons et évaluons plusieurs techniques pour améliorer les performances de JXTA sur une infrastructure réseau de type grille.

Mots clés : Pair-à-pair, calcul sur grille, JXTA, couches de communications, performance.

1 Using P2P techniques to build grids

Nowadays, scientific applications require more and more resources, Such as processors, storage devices, network links, etc. Grid computing provides an answer to this growing demand by aggregating processing and storage resources made available by various institutions. As their sizes grow, grids express an increasing need for flexible distributed mechanisms allowing them to be efficiently managed. Such properties are exhibited by peer-to-peer (P2P) systems, which have proven their ability to efficiently handle millions of interconnected resources in a decentralized way [38, 6]. Moreover, these systems support a high degree of resource volatility. The idea of using P2P approaches for grid resource management has therefore emerged quite naturally [11, 26].

To our knowledge, the very few practical attempts of convergence between P2P and grid computing have taken two different paths. One approach consists in implementing P2P services on top of software building blocks based on current grid technology (e.g., by using grid services as a communication layer [27]). Conversely, P2P libraries can be used on physical grid infrastructures as an underlying layer for higher-level grid services [1]. This is a way to leverage scalable P2P mechanisms for resource discovery, resource replication and fault tolerance. In this paper, we focus on this second approach.

Grid applications often have important performance constraints. In most cases, grids are built as cluster federations. System-Area Networks (SANs), such as Giga Ethernet or Myrinet (which typically provide Gb/s bandwidth and a few microseconds latency), are used for connecting nodes inside a given high-performance cluster; whereas Wide-Area Networks (WANs), provide a typical bandwidth of 1 Gb/s, but a higher latency (typically of the order of 10-20 ms), are used to connect the clusters. Consequently, sending a small message between two nodes within the same SAN may be 1,000 times less expensive then doing the same operation across a WAN. Such a discrepancy cannot be neglected, since the efficient use of the network characteristics is a crucial issue in the context of performance-constrained scientific applications.

In contrast, P2P applications generally do not have important performance constraints, as they usually target the edges of the Internet (with low-bandwidth and high-latency links, such as Digital Subscriber Line (DSL) connections). In such a context, the latency between arbitrary pairs of nodes does not exhibit a high variation. Therefore, most published papers on P2P systems generally model the communication cost as a distance based on the number of logical hops between the communicating entities, without taking into account the underlying physical topology. When running P2P protocols on grid infrastructures, this factor clearly has to be considered in order to efficiently use the capacities of the available networks to provide the performance required by the applications. Therefore, using P2P libraries on grid infrastructures as building blocks for grid services is a challenging problem,

since this is clearly an unusual deployment scenario for P2P systems. Consequently, it is important and legitimate to ask: are P2P communication mechanisms adequate for a usage in such a context? Is it possible to adapt P2P communication systems in order to benefit from the high potential offered by these high-performance networks, in order to meet the needs of scientific grid applications?

Most of the few attempts to realize the P2P-grid convergence have been based on the JXTA [34] open-source project (see the related work below). In its 2.0 version, JXTA consists of a specification of six language- and platform-independent, XML-based protocols that provide basic services common to most P2P applications, such as peer group organization, resource discovery, and inter-peer communication. This paper discusses the appropriateness of using the JXTA P2P platform for two classes of applications deployed on very different types of networks. First, we address classical collaborative computing usually deployed over Internet. For such applications, a study based on Fast Ethernet is appropriate. Second, we consider high-performance computing applications, running on grid infrastructures. This second case is more challenging for JXTA, as it evaluates to what extent its communication layers are able to leverage high-performance (i.e. Gigabit/s) networks. For the purpose of our evaluation, we use the well-known bidirectional bandwidth benchmark, widely used to evaluate networking protocols. This paper focuses on the evaluation of JXTA-J2SE and JXTA-C[1] over LANs and high-performance SANs and WANs. It also proposes and discusses several solutions to improve the raw performance observed.

The remainder of the paper is organized as follows. Section 2 introduces the related work: we discuss some JXTA-based attempts for using P2P mechanisms to build grid services and we mention some past performance evaluations of JXTA. Section 3 provides an overview of the communication layers of both JXTA-J2SE and JXTA-C. Section 4 describes in detail the experimental setup used for LAN, SAN and WAN benchmarks. Sections 5, 6 and 7 present the benchmark results of JXTA over these three types of networks. In Section 8, we discuss the measurements from a global perspective through a comparison with other middleware typically used for building grid applications. We also provide some hints on how to efficiently use JXTA's communication layers. Finally, Section 9 concludes the paper and discusses some possible future directions.

2 Related Work

Several projects have focused on the use of JXTA as a substrate for grid services. The Cog Kit JXTA Project [35] and the JXTA-Grid [36] project are two examples. However, none of these projects are being actively developed an longer and none has released any

[1] The only two bindings compliant to JXTA's specifications version 2.0

prototypes. The Service-oriented Peer-to-Peer Architecture [1] (SP2A) project aims at using P2P routing algorithms for publishing and discovering grid services. SP2A is based on two specifications: the Open Grid Service Infrastructure (OGSI) and JXTA. None of the projects above has published performance evaluations so far. Finally, JUXMEM [2] proposes to use JXTA in order to build a grid data-sharing service. All projects mentioned above share the idea of using JXTA as a low-level interaction substrate over a grid infrastructure. Such an approach brings forth the importance of JXTA's communications performance, which may be critical for scientific grid applications.

In this paper, we focus on the communication layers of the main two bindings of JXTA: JXTA-J2SE and JXTA-C. The performance of the widely-used *pipe* communication layer of JXTA-J2SE has been the subject to many studies [14, 15, 23, 24, 16, 25] and has served as reference for comparisons with other P2P systems [5, 18, 29]. However, these studies are primarily based on JXTA 1.0 [14, 15] or even older [23, 24]. Newer versions, starting with JXTA 2.0, have been claimed to introduce significant design enhancements making these results obsolete. Additionally, in all previously studies benchmarks are performed at the application-level without any in-depth analysis of the behavior of the communication layers. Besides, most of the benchmarks have been performed on hybrid JXTA virtual network configurations (involving several types of JXTA peers, simultaneous use of HTTP and TCP transport protocols, etc), which makes the understanding of the underlying costs very difficult. Consequently, no comprehensive discussion and explanation of the experimental results has been proposed so far. In other studies, some performance results with respect to latency performance are inconsistent (e.g. [25] on one side and [15, 16] on the other side). This makes it hard to get a clear view of the performance of JXTA communication layers.

The most recent evaluations of both main JXTA bindings are [3] and [4]. The main focus of [3] is a performance evaluation of JXTA-J2SE in a LAN environment using Fast Ethernet. This paper also provides the first evaluation of JXTA-C and gives hints on how to use both bindings of JXTA in order to get good performance on this kind of networks. In [4], the same benchmark code is used but in a very different context: performance evaluations are performed on grid infrastructures consisting of SAN-based clusters interconnected by high-bandwidth WANs. To the best of our knowledge, [4] is the first attempt to discuss the appropriateness of using the JXTA P2P platform for high-performance computing on grid infrastructures.

This paper presents an extended and updated version of the results preliminarily presented in [3] and [4], which are synthesized and discussed from a larger perspective. We include some new performance figures for SAN and WAN benchmarks, using the latest version of JXTA-C (2.1.1), which exhibits significant improvements compared to previous performance measurements published in [4]. In the context of the convergence of P2P and

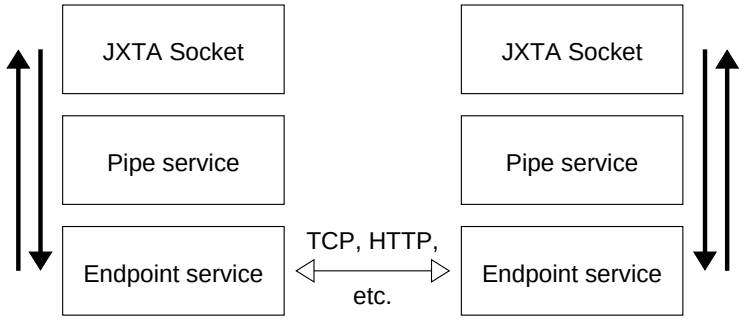


Figure 1: Stack of JXTA communication protocols.

grid computing, these updated results can help the user to draw a clear picture of the potential of JXTA-C on high-performance networks available on grid infrastructures.

3 Overview of JXTA Communications Layers

JXTA provides three basic transport mechanisms for inter-peer communication, each providing a different level of abstraction. The *endpoint service* is the lowest level transport mechanism, followed by the *pipe service*, and then finally, at the highest level, there are *JXTA sockets*. As shown in Figure 1, each transport mechanism is built on top of the transport mechanism below it. The endpoint service utilizes the available underlying transport protocols (for example TCP).

At the lowest level, information is exchanged between peers in discrete units known as *JXTA messages*. JXTA specifies two possible wire representations for a JXTA message: binary, where a transport protocol such as TCP is available; and XML, in case the underlying transport protocol is not capable of transmitting binary data. In either case, a JXTA message is comprised of a series of named and typed *message elements* [37], any number of which may be required by the transport protocol or added by the application as the message payload. These message elements can be of any type, including, for example, an XML document.

3.1 The bottom layer: the endpoint service

The endpoint service is JXTA's point-to-point communication layer. It provides an abstraction for the available underlying transport protocols (called *endpoints*) which can be used

to exchange data between one peer and another. Currently, supported transport protocols common to both implementations of JXTA are TCP and HTTP. However, regardless of the underlying transport protocol, all communications at the endpoint level, are asynchronous, unidirectional and unreliable.

Structure of an endpoint message. Using the interface provided by the endpoint service, all the information needed by a peer in order to send a message to another is the respective *endpoint address* of the corresponding destination peer. An endpoint address is basically just the JXTA virtual network address of the peer, also known as the Peer ID. The endpoint service then makes use of one of the JXTA protocols, namely the endpoint router protocol, to find an appropriate route to the destination peer using available transports and to resolve the underlying physical network address. When messages are exchanged between peers, two message elements, the `EndpointSourceAddress` and `EndpointDestinationAddress`, are used by the endpoint service to identify the origin and intended recipient peer of the message in transit. They contain information about the physical location of the peer on the network, such as the TCP address of the peer, and are required by the endpoint service to be present in all messages sent by this service. Additionally, the `EndpointDestinationAddress` message element specifies the name of the upper service in charge of handling the received message. This service can be for example a user-based service using the endpoint layer as its communication layer.

In general, the endpoint service should not be utilized directly by applications, but rather indirectly through the use of one of the upper communication layers, such as the pipe service or JXTA sockets. Therefore, the aim of benchmarking the endpoint service is primarily to gather performance data on the endpoint service for the purpose of explaining the performance measured for these upper layers.

3.2 The core communication layer: the pipe service

The pipe service supplements the endpoint service by incorporating the abstraction of virtual communication channels (or *pipes*). The aim of the pipe service is to provide the illusion of a virtual endpoint that is independent of any single peer location and network topology, as stipulated by JXTA specifications. Like peers, each pipe also has an identifier unique to the JXTA virtual network; this is known as the Pipe ID and is used by the pipe service to bind peers to *pipe-ends*. Before a message is transferred between peers, each end of the pipe is resolved to an endpoint address, through the use of JXTA's pipe binding protocol, and the endpoint service is used to handle the actual details of transferring messages between peers (the resolution is only done once for each pipe and is subsequently checked every 20 minutes in the JXTA-J2SE implementation).

Like endpoint communications, pipe communications are also asynchronous and unreliable. However, the pipe service offers two modes of communication: point-to-point mode, through the use of *unicast pipes*, and propagate mode, through *propagate pipes*. In propagate pipes, a single peer can simultaneously send data to many other peers. In point-to-point mode, it is also possible to exchange encrypted data through the use of *secure pipes*. However, in this study we focus on basic unicast pipes because of their general-purpose nature and because they serve as a basis for the implementation of the higher-level JXTA sockets.

Structure of a pipe message. In terms of message composition at the pipe service level, the service name inside the `EndpointDestinationAddress` message element is specified to be the endpoint router service. In addition to the message elements required by the endpoint service, an extra message element, the `EndpointRouterMsg` message, is also present in each message exchanged via the pipe service. This additional message element plays a role in the delivery of a JXTA message to applications using the pipe service, as it contains at this layer the ID of the pipe. Specifications also state that the `EndpointRouterMsg` message element is used by the endpoint router service to facilitate the routing of the message for peers that are unable to exchange messages directly over the network. However, this message element is included by the pipe layer even when a direct connection is available between peers.

3.3 Enabling sockets over P2P: JXTA Sockets

The JXTA sockets introduce yet another layer of abstraction on top of the pipes and provide an interface similar to that of the more familiar BSD socket API. Compared to the JXTA pipes, JXTA sockets add reliability and bi-directionality to JXTA communications. Additionally, JXTA sockets transparently handle the packaging and un-packaging of application-specific data into and out of JXTA messages, presenting a data-stream type of interface to each of the communicating peers. However, it should be noted that this layer is not part of the core specifications of JXTA and has not been implemented in JXTA-C so far. It was introduced in JXTA-J2SE 2.0, with reliability support added in 2.1.

Structure of a JXTA socket message. JXTA sockets add another message element beyond those required by the pipe service: the `ACK_NUMBER` message element. From the user perspective, the `ACK_NUMBER` is the most important message element since it encapsulates the actual message payload and some additional data used by the JXTA socket to ensure message reliability and proper message sequencing at the destination peer.

The data-stream interface also introduces another interesting parameter which can be used to tune JXTA sockets. Indeed, it is possible to configure the size of the output buffer of a JXTA socket. This value has an impact on the way the socket packages the data it receives for transmission into a series of separate JXTA messages that can be sent using the pipe service. This is significant because the JXTA socket creates a new JXTA message every time the buffer becomes full or the buffer is explicitly flushed by the application. In all versions of JXTA, the default buffer size is 16 KB.

4 Description of the Experimental Setup

For all reported measurements we use a *bidirectional bandwidth* benchmark (between two peers), based on five subsequent time measurements of an exchange of 100 consecutive message-acknowledgment pairs sampled at the application level. We chose this test as it is a well-established metric for benchmarking networking protocols, and because of its ability to yield information about important performance characteristics such as bandwidth and latency. Moreover, such information is necessary in order to evaluate JXTA's adequacy for performance-constrained grid applications. Both bindings of JXTA were configured to use TCP as the underlying transport protocol.

When benchmarks are performed using JXTA-J2SE, the Sun Microsystems Java Virtual Machine (JVM) 1.4.2 is used and executed with `-server -Xms256M -Xmx256M` options. The JVM 1.4.1 from IBM is also used for some experiments, with the following options: `-Xms256M -Xmx256M`; such uses of the IBM JVM are explicitly stated in the performance analysis. It should be noted that JXTA 2.2.1 does not run on top of the IBM JVM because of the use of `javax` classes in a required library of JXTA 2.2.1 (these classes are not supported by IBM's JVM). Also note that when the J2SE binding of JXTA is benchmarked, an additional warm-up phase based on 1000 consecutive message-acknowledgment pairs is performed, to make sure that the Just-In-Time (JIT) compiler is not disturbing the measurements. The JXTA-C benchmarks are compiled using `gcc` 3.3.3 for the LAN benchmarks and `gcc` 4.0 for the SAN and WAN benchmarks. In both cases, the `O2` level of optimization is used. Finally, note that all source codes required to perform the benchmarking of each communication layer of JXTA-J2SE and JXTA-C have been made available via the web sites of the JDF [7] and JXTA-C [39] projects.

Protocol efficiency is the other factor explored in the performance evaluation of the JXTA protocols. This measure is defined as the ratio between the amount of data that a user wishes to send and the total amount of data actually required by the protocol to send it. Therefore, any additional data included in the transmission of the message payload will ultimately reduce the efficiency of the protocol and may inhibit performance. Results are

given by analyzing exchanged messages between peers through the use of two network protocol analyzers: *tcpdump* and *ethereal*.

As seen in section 1, a clear picture with respect to the performance characteristics of JXTA communication layers is necessary before attempting to use it in the development of any specific P2P service. However, P2P applications target various kinds of uses, ranging from classical collaborative platform and distributed computing over Internet, such as JXCube [33] and P3 [25] respectively, to data-sharing services over grid infrastructures, such as JUXMEM [2]. These various use cases motivate the need for benchmarking and analyzing JXTA communication layers on top of different networks: LANs and high-performance SANs and WANs. The last two are typically used within grid infrastructures. However, note that due to network upgrades in the testbed used, the software versions of JXTA used for the LAN benchmarks are slightly older than the ones used for SAN and WAN benchmarks.

LAN benchmarks. The network used for the LAN benchmarks is Fast Ethernet (100 Mb/s). Nodes consist of machines using 2.4 GHz Intel Pentium IV processors, outfitted with 1 GB of RAM each, and running a 2.4 version Linux kernel. Tests were executed using version 2.2.1 (released the 15th of March 2004) and 2.3 (released the 15th of June 2004) for the J2SE binding. For the C binding, the CVS head of JXTA-C from the 8th of November 2004 was used (the only modification was the deactivation of Nagle's algorithm in the TCP protocol). Since direct communication amongst nodes of LAN-based cluster is usually available, direct communications between peers has also been configured. Note that this is allowed by JXTA specifications.

SAN benchmarks. The networks used for the SAN benchmarks are Giga Ethernet and Myrinet (GM driver, version 2.0.11). When the network layer is Myrinet, nodes consist of machines using 2.4 GHz Intel Pentium IV processors, outfitted with 1 GB of RAM each, and running a 2.4 version Linux kernel. For Giga Ethernet, nodes consist of machines using dual 2.2 GHz AMD Opteron processors, also outfitted with 1 GB of RAM each, and running a 2.6 version Linux kernel. Benchmarks were executed using versions 2.2.1 and 2.3.2 of the J2SE binding of JXTA. For the C binding, version 2.1.1 (released the 14th of June 2005) was used. As for the LAN benchmarks, since direct communication among nodes of a SAN-based cluster is usually available in the currently deployed grids, direct communications between peers has also been configured.

WAN benchmarks. The platform used for the WAN benchmarks is the Grid'5000 French national grid platform [32]. Tests were performed between two of the Grid'5000 clusters located in Rennes and Toulouse. On each side, nodes consist of machines using

dual 2.2 GHz AMD Opteron processors, outfitted with 1 GB of RAM each, and running a 2.6 version Linux kernel. The two sites are interconnected through a 1 Gb/s link, with an average measured latency of 11.2 ms. As for the SAN benchmarks, benchmarks were executed using the versions 2.2.1 and 2.3.2 of the J2SE binding of JXTA. For the C binding, the version 2.1.1 (release 14th of June 2005) was used. In this case too, we configured JXTA peers to enable direct exchanges. As stated previously, this is clearly an unusual deployment scenario for P2P systems, in which direct communication between peers is the generally rather exception than the rule (because of firewalls, etc.). However, let us stress that some grids provide direct communication only between cluster front-ends. To cope with that case, additional evaluations would be necessary.

5 Performance Evaluation of JXTA over Local-Area Networks

This section presents an analysis of the results obtained for the performance of JXTA's communications layers on LAN. On this configuration, the bandwidth of plain sockets is around 11.2 MB/s and 70 μs respectively (average values between C and Java sockets). These values are used as a reference performance bound.

5.1 Analysis of JXTA-J2SE's Performance

JXTA-J2SE endpoint service. Figures 2 and 3 show that the endpoint service of JXTA 2.2.1 and 2.3 nearly reach the bandwidth of plain sockets over LAN networks: 11.20 MB/s for JXTA 2.2.1 and 11.15 MB/s for JXTA 2.3. However, the results shown on these figures were obtained by modifying the source code of JXTA. A bound is indeed enforced by JXTA-J2SE on the size of messages. This limitation was introduced into JXTA to promote some fairness in resource sharing among peers on the network, for instance when messages must be stored on relay peers (the type of peer required to exchange messages through firewalls). However, between JXTA 2.2.1 and JXTA 2.3, this bound was lowered from 512 KB to 128 KB (of application-level payload). Note that the endpoint service of JXTA 2.2.1 achieves slightly better results up to the bound imposed on the size of messages for this version of JXTA. The default peak bandwidth of JXTA 2.3 is only 10.47 MB/s, whereas for JXTA 2.2.1 it is 11.01 MB/s. On the latency side and as shown on table 1, the endpoint service for both versions of JXTA achieve latency measurements in the sub-millisecond range: 960 μs for JXTA 2.2.1 and 735 μs for JXTA 2.3; with the IBM JVM, JXTA 2.3 even achieves a latency of 485 μs. Additionally, the latency is also affected by transmitting two TCP packets for every JXTA message (this has since been fixed in JXTA 2.3.1). Still, when using the IBM JVM, the latency only improves by 16 μs. Note

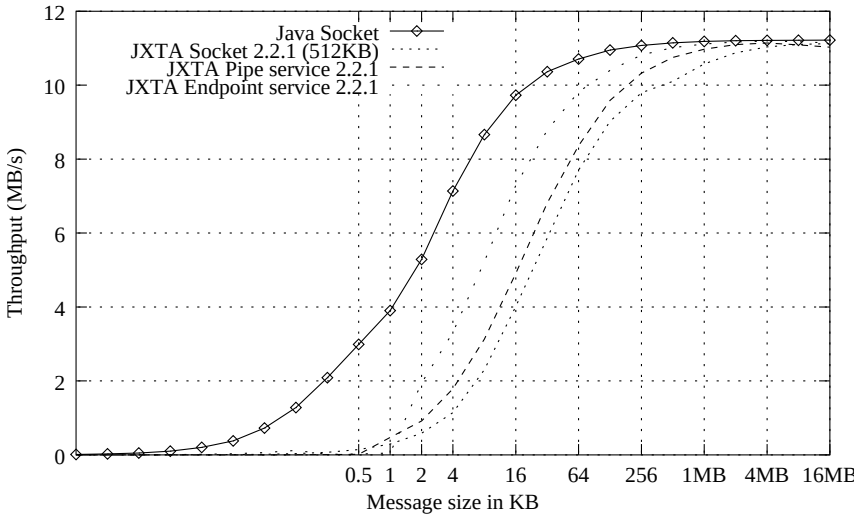


Figure 2: Bandwidth of each layer for JXTA-J2SE 2.2.1 compared to Java sockets.

that the protocol efficiency of JXTA's endpoint layer is: 300 bytes for a 1-byte message payload.

JXTA-J2SE unicast pipe. As in the case of the endpoint layer, Figures 2 and 3 illustrate the similarity of the shapes of the JXTA unicast pipe versions 2.2.1 and 2.3. Again, the message size limit prevents JXTA 2.3 bandwidth (9.59 MB/s) from reaching the higher bandwidth of JXTA 2.2.1 (10.74 MB/s). By removing this limit, the peak bandwidth of JXTA 2.2.1 increases to 11.14 MB/s for message size of above 1 MB. This is a significant improvement (14%) over the peak bandwidth exhibited by the default unicast pipes limited to message sizes of 512 KB. Although the bandwidth results are similar, there are some noteworthy differences in latency measurements between JXTA 2.2.1 and 2.3, as shown on table 1. JXTA 2.3 yields a latency of around 2 ms, while the latency of JXTA 2.2.1 is 35 ms. Note that when the IBM JVM is used, the latency of JXTA 2.3 goes down even further to 1.3 ms. The deactivation of the TCP packet aggregation mechanism in JXTA 2.3 explains the discrepancy of latency results. The buffering is therefore now explicitly performed within the endpoint layer of JXTA-J2SE, allowing one TCP packet to be sent for a single JXTA message with a minimal latency. The higher latency measured for JXTA 2.3, compared to the latency of the endpoint layer, is explained by the additional message element added by the pipe service into the JXTA message. As described in Section 3, the pipe

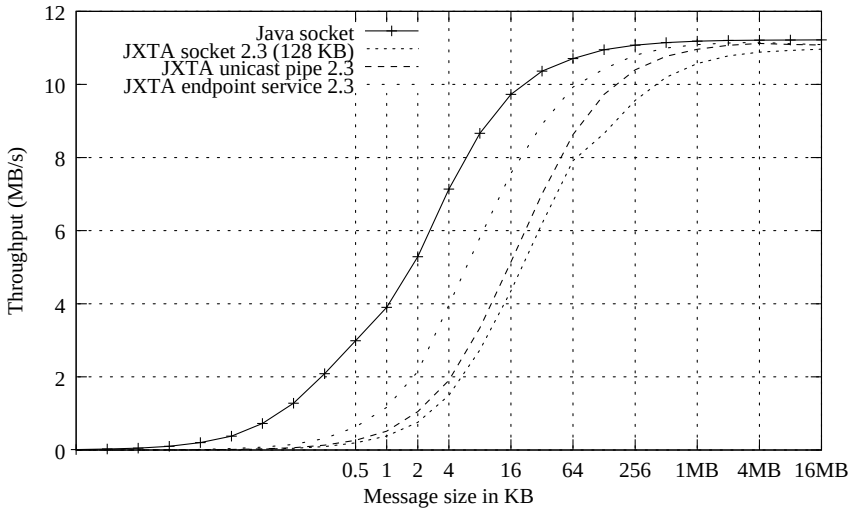


Figure 3: Bandwidth of each layer for JXTA-J2SE 2.3 compared to Java sockets.

Version of JXTA-J2SE	2.2.1	2.3
Endpoint service	960 μs	480 μs
Unicast pipe	35 ms	2 ms
JXTA socket	3.4 ms	2.5 ms

Table 1: Latency results for JXTA-J2SE 2.2.1 and 2.3 on LAN.

service layer introduces a message element called the `EndpointRouterMsg`. As the `EndpointRouterMsg` is an XML document, the costly parsing required to process this element explains the higher latency observed. Moreover, its size (565 bytes) contributes to the very poor protocol efficiency of unicast pipes compared to the endpoint service: the total message size for a 1-byte message payload is 877 bytes (the protocol efficiency decreases to less than half that of the endpoint layer).

JXTA-J2SE sockets. Figures 2 and 3 show that the performance of JXTA sockets 2.2.1 and 2.3 achieve a peak bandwidth of 11.12 MB/s and 10.96 MB/s. However, note that these results were obtained by setting the output buffer to 512 KB for JXTA sockets 2.3 and to 128 KB for JXTA sockets 2.2.1. Note that, in their default configuration,

JXTA sockets 2.2.1 and 2.3 reach maximum bandwidths of only 9.48 MB/s and 9.72 MB/s, respectively. These low bandwidths, compared to plain sockets are, explained by the default output buffer size of JXTA sockets, 16 KB. Any messages much larger than the size of this output buffer must be fragmented into several hundred smaller messages before transmission, resulting in reduced performance. On the latency side, JXTA sockets reach around 3.4 ms for the 2.2.1 version and 2.5 ms for the 2.3 version, as shown on table 1. When the IBM JVM is used, the latency is reduced to 1.76 ms for JXTA 2.3. The only difference between JXTA sockets messages and pipe messages is that the `PAYLOAD` element is replaced by the `ACK_NUMBER` element, a message that still contains the application-level payload but with extra data to guarantee reliability. This additional data and the extra processing required in order to achieve reliable communications explains the higher latency of JXTA sockets as compared to unicast pipes. Furthermore, this message element slightly decreases the protocol efficiency of JXTA sockets compared to that of unicast pipes: for a 1-byte message payload, the total size of the JXTA message that is actually transferred is 913 bytes.

Discussion. In conclusion, the measurements show that the two main JXTA transport mechanisms directly used by JXTA-based applications (unicast pipe and JXTA sockets) are both able to reach the bandwidth of plain sockets on a Fast Ethernet local-area network. Moreover, Figures 2 and 3 show that the performance difference between JXTA sockets and unicast pipes for sending large messages is negligible. Therefore, using JXTA sockets is recommended as it offers reliability guarantees. Table 1 shows that, on the latency side, the two main transport mechanisms exhibit poor performance. This is explained by the costly processing of the rather large XML document included in each message. Without this element, the endpoint service reduces the gap in latency between the JXTA protocols and Java sockets but is still 400 μs higher. This overhead has not been explained so far; however, we suspect they are due to inappropriate thread management (scheduling/creation/description), as tests have demonstrated that the number of used threads may vary between 33 and 40. It is also interesting to note that latency improvements have been observed when upgrading the Linux kernel from 2.4 to 2.6, probably thanks to the new thread library of Linux kernel 2.6. For example, the latency results of the endpoint layer has improved by around 70 μs for JXTA-J2SE 2.3.

5.2 Analysis of JXTA-C's Performance

JXTA-C endpoint service. Figure 4 shows that the endpoint service for JXTA-C achieves a peak bandwidth of 11.16 MB/s, to be compared to 11.7 MB/s for plain sockets. Note that no limit on the message size is currently implemented in JXTA-C. On the

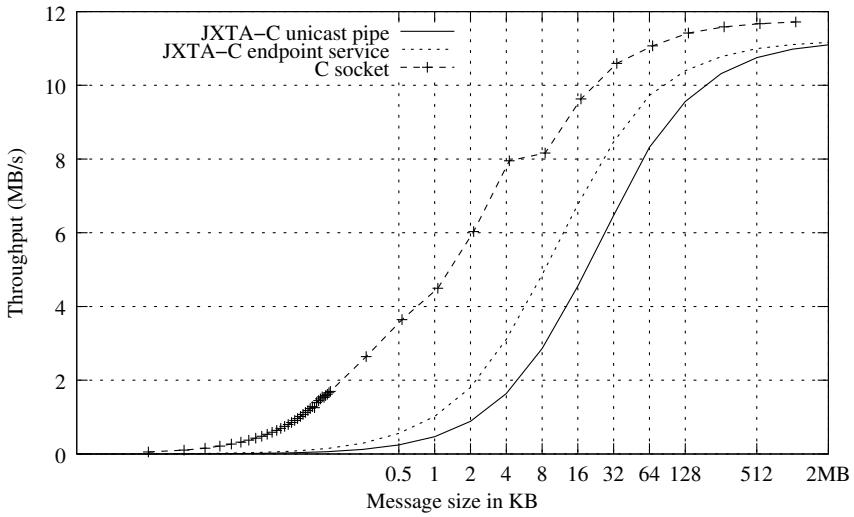


Figure 4: Bandwidth of each layer for JXTA-C compared to C sockets.

Endpoint service	0.82 ms
Unicast pipe	1.99 ms

Table 2: Latency results for JXTA-C on LAN.

latency side, JXTA-C endpoint service exhibits results around 820 μs, as shown in Table 2. The difference observed as regards the latencies measured for the J2SE and C bindings of JXTA is mainly due to the lack of buffering in the endpoint layer for the C binding. As we disabled the TCP packet aggregation algorithm to reduce latency, an explicit buffering scheme in the endpoint layer is required so that a single TCP packet be sent for each JXTA message, with a minimal latency, as it was done in JXTA-J2SE. Implementing this mechanism is expected to significantly improve latency results of JXTA-C (this issue has been solved by that time and is available since JXTA-C 2.1). The protocol efficiency of the endpoint service is slightly better than its J2SE counterpart: 239 bytes are sent for a 1-byte application payload (instead of 300 bytes). This is due to the fact that the JXTA-J2SE version systematically specifies the encoding tags, whereas the JXTA-C version omits them when default values are used.

JXTA-C pipe service. Figure 4 also shows that the peak bandwidth of unicast pipes is 11.1 MB/s compared to 11.7 MB/s for plain sockets. As illustrated in Table 2, the pipe latency for JXTA-C is around 2 ms, much higher than the latency observed for the endpoint service. As for JXTA-J2SE, these results are explained by the composition of a message which is identical to its J2SE counterpart. Therefore, the same conclusion applies: the presence of the `EndpointRouterMsg` adds a costly XML-parsing step. However, the efficiency is slightly better than in the case of JXTA-J2SE: for a 1-byte message payload, the total size of the JXTA message that is actually transferred is 834 bytes (instead of 877 bytes).

Discussion. JXTA-C is a reviving project, developed by fewer people and consequently struggling to reach JXTA-J2SE's level of features. Therefore, this analysis is somewhat shorter than its J2SE counterpart. Overall, similarly to the J2SE binding of JXTA, the C binding of JXTA is able to reach the bandwidth of plain sockets on a Fast Ethernet local-area network. Note that the JXTA socket layer is not implemented in the C binding of JXTA. However, as regards latency, JXTA-C exhibits poor results due to the processing of a large XML document at the pipe layer and to an unsatisfactory buffering mechanism at the endpoint layer. These issues as well as unoptimized code in the communications layers prevents JXTA-C from reaching latency results close to plain socket one. However, note that the unsatisfactory buffering mechanism has been improved since version 2.1 of JXTA-C (released the 15th of March 2005). Therefore, let us stress again that results shown by SAN benchmarks in the next section take into account this improvement.

6 Performance Evaluation of JXTA over System-Area Networks

This section analyzes the performance of JXTA's communications layers on SANs. Note that for Myrinet, the *Ethernet emulation* mode of GM 2.0.11 is used and configured with jumbo frames. This mode allows Myrinet to carry any packet traffic and protocols that can be transported by Ethernet, including TCP/IP. Although this capability is bought at the cost of losing the main advantage of a Myrinet network (the OS-bypass mode), it allows the same socket-based benchmarks to be run unmodified. On this configuration, the bandwidth and latency of plain sockets is around 155 MB/s and 60 μs respectively, whereas on Giga Ethernet it is around 115 MB/s for the bandwidth and 45 μs for the latency (average values between C and Java sockets). As in the case of the LAN benchmarks, these values are used as a reference performance bound.

Version of JXTA	JXTA-J2SE 2.2.1		JXTA-J2SE 2.3.2	
Network	Myrinet	Giga Ethernet	Myrinet	Giga Ethernet
Endpoint service	890 μs	357 μs	624 μs	294 μs
Unicast pipe	1.9 ms	834 μs	1.7 ms	711 μs
JXTA socket	3.3 ms	1.3 ms	2.4 ms	977 μs

Table 3: Latency results for JXTA-J2SE 2.2.1 and 2.3.2 on SAN.

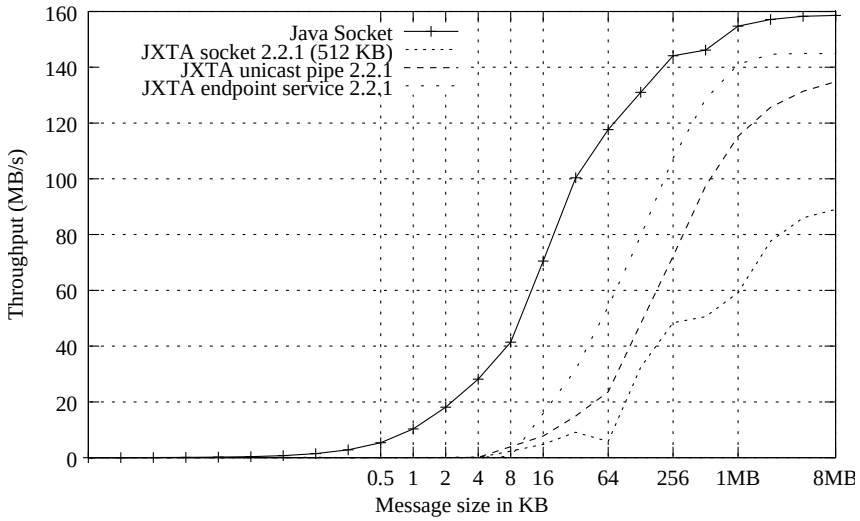


Figure 5: Bandwidth of each layer of JXTA-J2SE 2.2.1 as compared to Java sockets over a Myrinet network.

6.1 Analysis of JXTA-J2SE's Performance

JXTA-J2SE endpoint service. Figures 5 and 6 show that the endpoint service of JXTA 2.2.1 nearly reaches the bandwidth of plain sockets over SAN networks: 145 MB/s over Myrinet and 101 MB/s over Giga Ethernet. However, Figures 7 and 8 also show that the bandwidth of the JXTA 2.3.2 endpoint layer has decreased: drops of 32 MB/s over Myrinet and 20 MB/s over Giga Ethernet are observed, compared to the corresponding values measured for the 2.2.1 release. These lower bandwidths affect all versions of JXTA above its release 2.2.1 and are explained by a new implementation of the endpoint layer that shipped with JXTA 2.3. The profiling of JXTA has pointed out that this drop of performance is due to the mechanism used for limiting the size of messages sent by the endpoint layer,

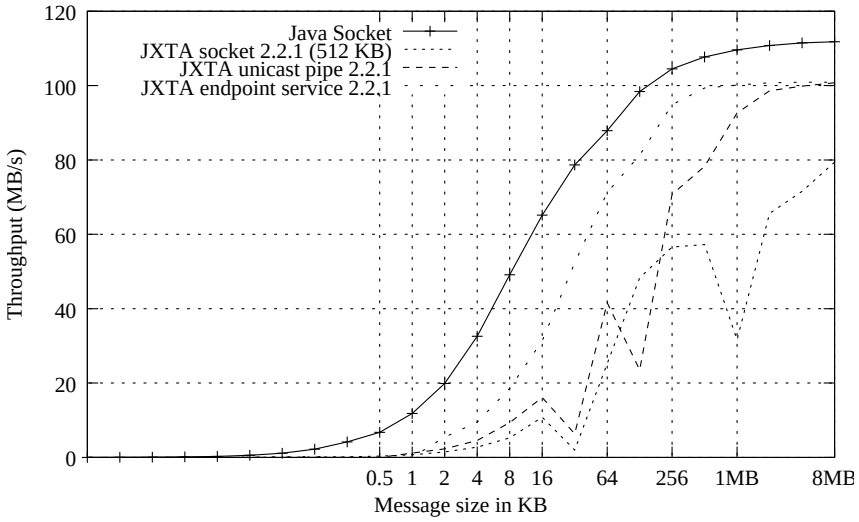


Figure 6: Bandwidth of each layer of JXTA-J2SE 2.2.1 as compared to Java sockets over a Giga Ethernet network.

described in section 5. As for the LAN benchmarks and since no relay peers are needed for the SAN benchmarks (see section 4), we removed this limit for our measurements. Table 3 shows that latency results of JXTA-J2SE have improved since version 2.2.1. The latency of the JXTA 2.3.2 endpoint service over Giga Ethernet reaches a value under 300 μs. Moreover, it goes down even further to 268 μs and 229 μs when using the SUN 1.5 and IBM 1.4.1 JVMs, respectively. Note that, the difference between Myrinet and Giga Ethernet results is due to the hardware employed, as the Ethernet emulation mode is used for Myrinet. Finally, also note that these improved results on the latency side, compared to the one of LAN benchmarks, are explained by the different characteristics of the network hardware used, as described in section 4.

JXTA-J2SE unicast pipe. In addition, Figures 5, 6, 7 and 8 also demonstrate a bandwidth degradation for JXTA-J2SE unicast pipes. For example, while JXTA 2.2.1 unicast pipe attains a good peak bandwidth of 136.8 MB/s over Myrinet, its 2.3.2 counterpart reaches a bandwidth of only 106.5 MB/s. A similar performance degradation can be observed on Giga Ethernet. However, the shape of the curve of unicast pipes 2.2.1 on Giga Ethernet has not been explained so far. We suspect the first drop is due to a scheduling problem, as discussed in Section 5.1. On the other hand, the reason of the drop at 128 KB

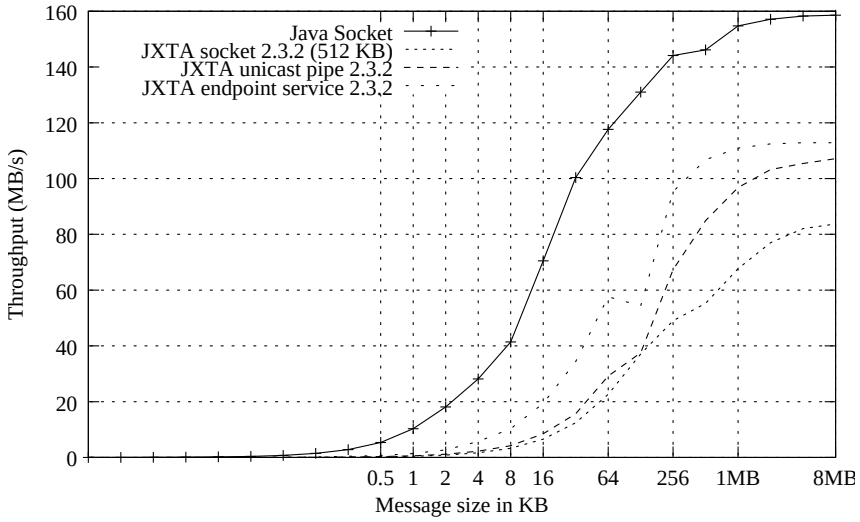


Figure 7: Bandwidth of each layer of JXTA-J2SE 2.3.2 as compared to Java sockets over a Myrinet network.

of application payload is still unknown. At the same payload size, a smaller drop for JXTA unicast pipes 2.3.2 over Giga Ethernet can be observed, but no link has been established with the previously mentioned drop, as this drop also occurs at the endpoint level. Overall, the small performance degradation as compared to the endpoint layer is explained by the composition of a pipe message: the presence of an XML message element requiring a costly parsing prevents this layer from reaching the performance of the endpoint layer. Moreover, as shown on Table 3, this extra parsing required for each pipe message also affects latency results: compared to the endpoint layer, latencies increase by more than 400 μs. However, unicast pipes are still able to achieve latencies in the sub-millisecond range, at least on Giga Ethernet.

JXTA-J2SE sockets. As opposed to the lower layers, JXTA sockets are far from reaching the performance of plain Java sockets. In their default configuration (e.g. with an output buffer size of 16 KB), JXTA sockets 2.2.1, for instance, attain a peak bandwidth of 12 MB/s over a Myrinet network. We were able to significantly improve the bandwidth and achieve 92 MB/s by increasing the size of the output buffer to 512 KB, as shown on Figure 9. Similar results were obtained over Giga Ethernet (for the sake of clarity, they are not represented on the Figure). Figure 9 also clearly shows the performance degradation be-

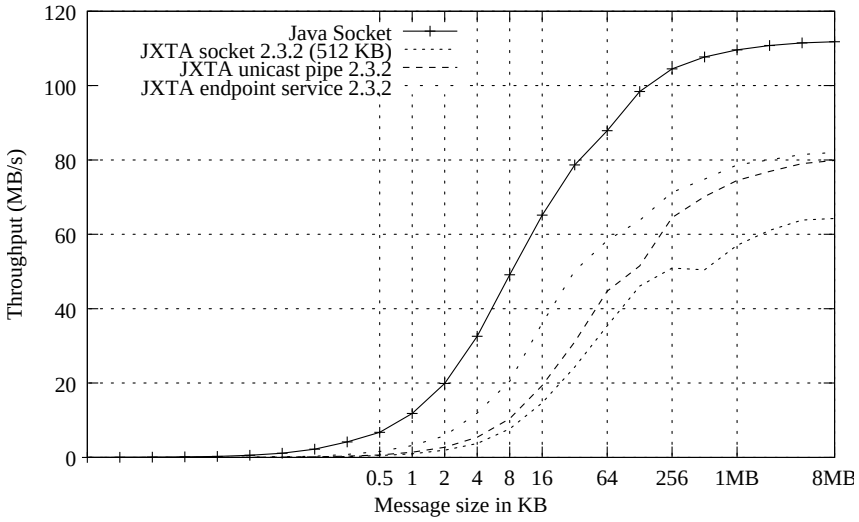


Figure 8: Bandwidth of each layer of JXTA-J2SE 2.3.2 as compared to Java sockets over a Giga Ethernet network.

tween JXTA sockets 2.2.1 and 2.3.2. As for the unicast pipes, the irregular shape of JXTA sockets 2.2.1 curves has not been explained so far. Again, we suspect the first drop is due to an inefficient thread scheduling policy. The next drop, when the message size is about the size of the output buffer, seems to be due to bugs discovered in the reliability layer since JXTA-J2SE 2.3.2. To the best of our knowledge, some progress has been made on these issues even if it can still be observed in JXTA 2.3.3. Table 3 highlights the progress being made by JXTA as regards latency, as only JXTA Sockets 2.3.2 on Giga Ethernet is able to reach a latency under one millisecond.

Discussion. In conclusion, JXTA-J2SE 2.2.1 communication layers are able to nearly saturate SANs, but only at the endpoint and pipe levels. The measurements revealed that the bandwidth of JXTA 2.2.1 is higher than JXTA 2.3.x. Latency results have largely improved since JXTA 2.2.1, but without reaching reasonably good performance for SANs. Finally, this evaluation has also highlighted that, in their default configuration, JXTA sockets achieve a very poor bandwidth. However, this result can significantly be improved by increasing the output buffer size. This requires the JXTA socket programmer to explicitly set this parameter in the user code. Based on these results, we can conclude that JXTA-J2SE can

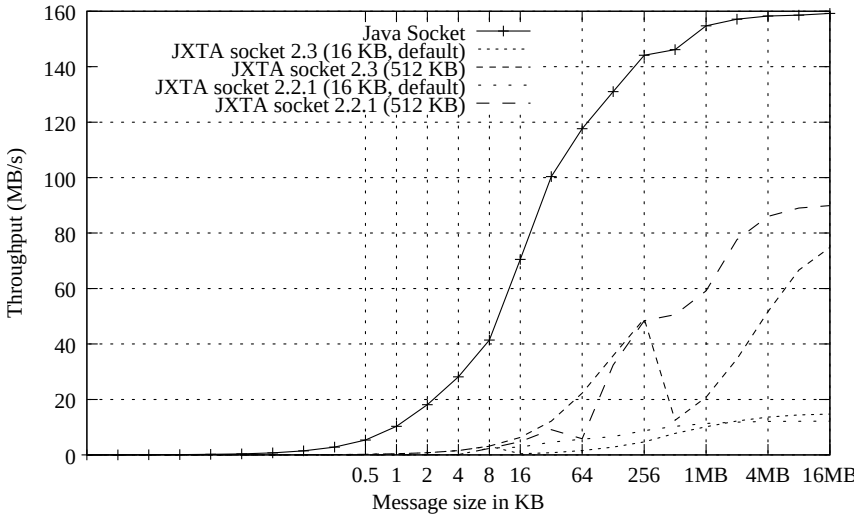


Figure 9: Bandwidth of JXTA-J2SE sockets 2.2.1 and 2.3.2 over a Myrinet network for two different output buffer sizes, compared to Java sockets.

be adapted in order to benefit from the potential offered by SANs, at least as bandwidth is concerned.

6.2 Analysis of JXTA-C's Performance

Figures 10 and 11 show the bandwidth measurements of all the communications layers of JXTA-C over SANs. Note that, as in the previous section, C sockets are used as an upper reference bound. The peak bandwidth values we measured for the endpoint service over Myrinet and Giga Ethernet are 135 MB/s and 103 MB/s respectively. The upper layer (unicast pipe) reaches bandwidths of 133 MB/s and 96 MB/s over Myrinet and Giga Ethernet, respectively. On Myrinet, this is an increase by 30 MB/s compared to results published in [4]. This highlights the improvements that have been made since JXTA-C 2.1. This is explained by the growing interest in using JXTA-C for not only research projects but also industrial projects. Therefore, JXTA-C now outperforms JXTA-J2SE 2.3.2 and achieves similar results compared to JXTA-J2SE 2.2.1. Moreover, these improvements have also largely improved the latency results compared to results of JXTA-C 2.1, as shown on Table 4. However, the latencies are still higher than the latency of plain sockets over Giga

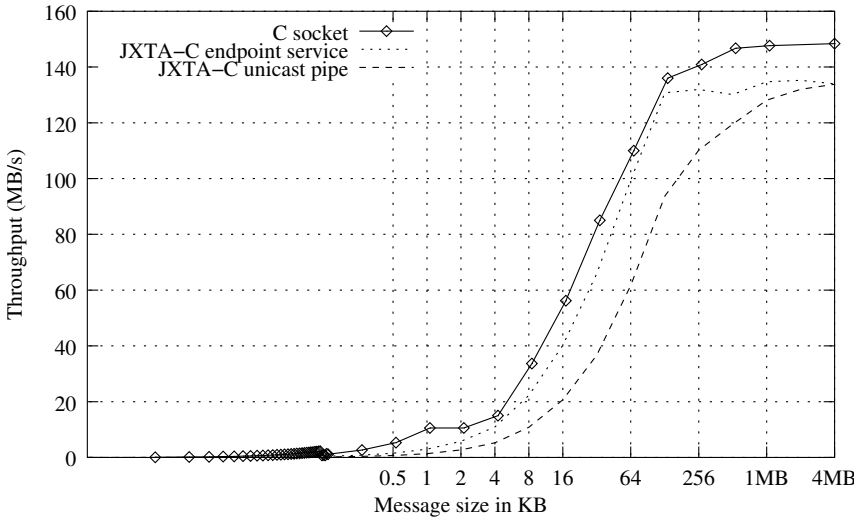


Figure 10: Bandwidth of each layer of JXTA-C 2.1.1 as compared to C sockets over a Myrinet network.

Version of JXTA	JXTA-C 2.1		JXTA-C 2.1.1	
Network	Myrinet	Giga Ethernet	Myrinet	Giga Ethernet
Endpoint service	635 μs	322 μ	310 μs	137 μs
Unicast pipe	1.7 ms	727 μs	690 μs	298 μs

Table 4: Latency results for JXTA-C 2.1 and 2.1.1 on SAN.

Ethernet (39 μs) (true especially for the pipe service). As for JXTA-J2SE, this is mainly explained by the unoptimized path a message must take in order to be sent on the wire, as well as unoptimized message composition when direct communications are available.

Based on this evaluation, we can conclude that, in their current implementation, the communication layers of JXTA-C are nearly able to saturate SANs, by reaching bandwidths values above 1 Gb/s. However, let us note that on the latency side, even if improvements have made throughout JXTA-C releases, results are far from efficiently using SAN capacities.

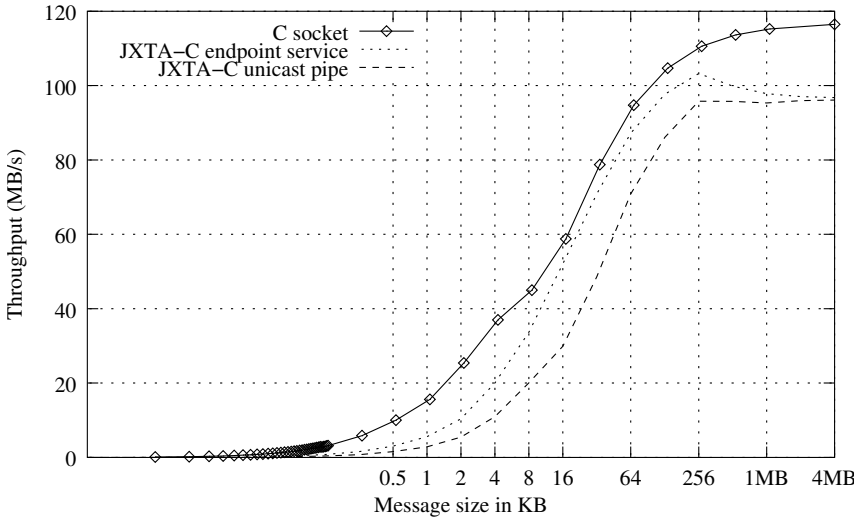


Figure 11: Bandwidth of each layer of JXTA-C 2.1.1 as compared to C sockets over a Giga Ethernet network.

6.3 Fully exploiting SAN capacities

In all evaluations we made using Myrinet, the Ethernet emulation mode of GM is used. However, this removes the ability to by-pass the IP stack of the OS and introduces an unneeded overhead. Consequently, communication layers are unable to fully exploit the capacities offered by Myrinet: full-duplex bandwidths of nearly 2 Gb/s and latencies below 7 μs thanks to zero-copy communication protocols. PadicoTM [10] is a high-performance framework for networking and multi-threading which allows middleware systems to transparently take advantage of such features. In this section, we focus on the *virtual sockets* feature offered by PadicoTM, which provides a way to directly access GM network interfaces. This is achieved by dynamically mapping, at runtime, standard socket functions on top of GM API functions, without going through the TCP/IP stack. Zero-copy is therefore possible and allows plain sockets to transparently reach a bandwidth of more than 230 MB/s and latency of 15 μs on Myrinet. This is a significant improvement compared to the basic performance achieved without PadicoTM (160 MB/s and 51 μs).

We have successfully ported JXTA-C to PadicoTM[2], without changing a single line of code of JXTA-C. We only performed a few modifications inside the OS-independent layer used by JXTA-C: the Apache Portable Runtime (APR). Basically, we replaced the default Posix thread library on which APR is based, by the Marcel [8] thread library used by PadicoTM. However, these modifications could be automatically achieved by a single `sed` command. The resulting peak bandwidth of JXTA-C's endpoint layer is 140 MB/s, thus reaching over 1 Gb/s. On the latency side, no significant improvements have been observed, as the non-optimized communication layers prevent JXTA-C from fully benefiting from the OS-bypass feature of Myrinet. Note that we did not use PadicoTM with JXTA-J2SE, since PadicoTM currently supports only the open-source Kaffe JVM. Unfortunately, this JVM is not compliant with Java specification version 1.4 and, therefore, is unable to run the Java binding of JXTA.

In conclusion, our experiments with PadicoTM show that JXTA could fully benefit from the potential performance of SAN networks if: 1) the implementation of all JXTA communication layers were carefully optimized, especially when direct communications are available 2) PadicoTM added support for JXTA-compatible JVMs (e.g. compliant to version 1.4 of Java's specifications). However, we believe that these issues will be solved in the near future.

7 Performance Evaluation of JXTA over Wide-Area Networks

This section analyzes the performance of JXTA's communications layers on WANs. Note that we had to tune the network settings of the nodes used for this benchmark. Our default maximum TCP buffer size initially set to 131072 bytes was limiting the bandwidth to only 7 MB/s. Based on the $bandwidth * delay$ law, we computed a theoretical maximum size of 1507328 bytes and increased this value by an arbitrary factor of 1.2. Therefore, we set the maximum TCP buffer sizes on each node to 1959526 bytes; `ttcp` configured with this value measured a raw TCP bandwidth of 107 MB/s, a reasonable level of performance.

JXTA-J2SE's performances. As for Giga Ethernet SAN benchmarks, Figures 12 and 13 show that the endpoint layer and unicast pipes of JXTA-J2SE are able to perform similarly to plain sockets over a high-bandwidth WAN of 1 Gb/s. This level of performance was reached by modifying JXTA-J2SE's code in order to properly set the TCP buffer sizes

[2]Note that due to time constraints the results presented in this section are based on the CVS head of JXTA-C from the 18th January 2005. However the final version of this paper will include results based on JXTA-C 2.1.1 as in the other sections.

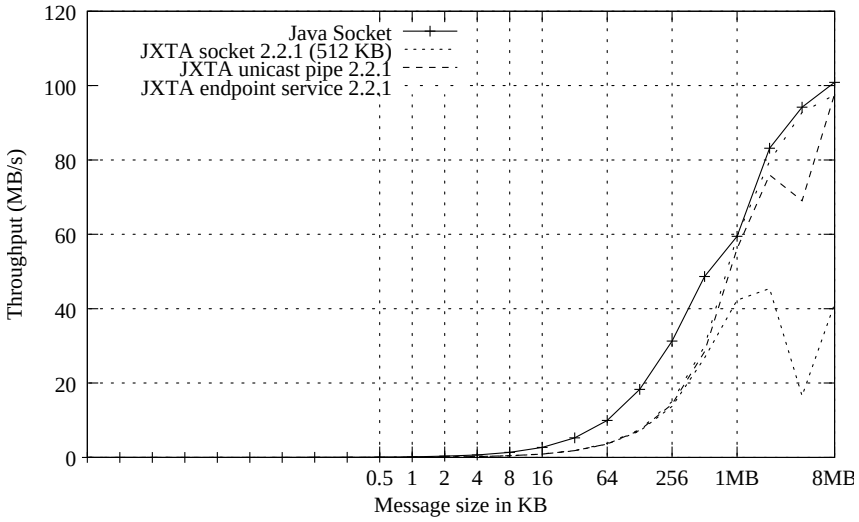


Figure 12: Bandwidth of each layer for JXTA-J2SE 2.2.1 compared to Java sockets over a high-bandwidth WAN.

to 1959526 bytes before binding sockets on both sides. Using the default setting, a bandwidth of only 6 MB/s was reached for JXTA 2.2.1 and less than 1 MB/s for JXTA 2.3.2. As opposed to SAN benchmarks, both versions of JXTA-J2SE achieve the same performance. This can be explained by the the fact that the higher latency of WANs hides the cost of the mechanism implemented for limiting the size of JXTA messages. Figures 12 and 13 also point out the same performance degradation for JXTA sockets as for SAN benchmarks. However, JXTA socket 2.3.2 achieves a higher bandwidth compared to its 2.2.1 counterpart. The performance drops of unicast pipes and JXTA sockets for JXTA-J2SE 2.2.1 for message size of 4 MB have not been explained so far.

JXTA-C's performances. Figure 14 shows that the peak bandwidth for both communication layers of JXTA-C 2.1.1 over WANs are 94 MB/s, for a message size of 4 MB. As for the the SAN benchmarks, these higher results compared to results published in [4] are explained by the improvements made since JXTA-C 2.1. However, this level of performance was only reached by modifying JXTA-C's code in order to properly set the TCP buffer sizes to 1959526 bytes before binding sockets on both sides.

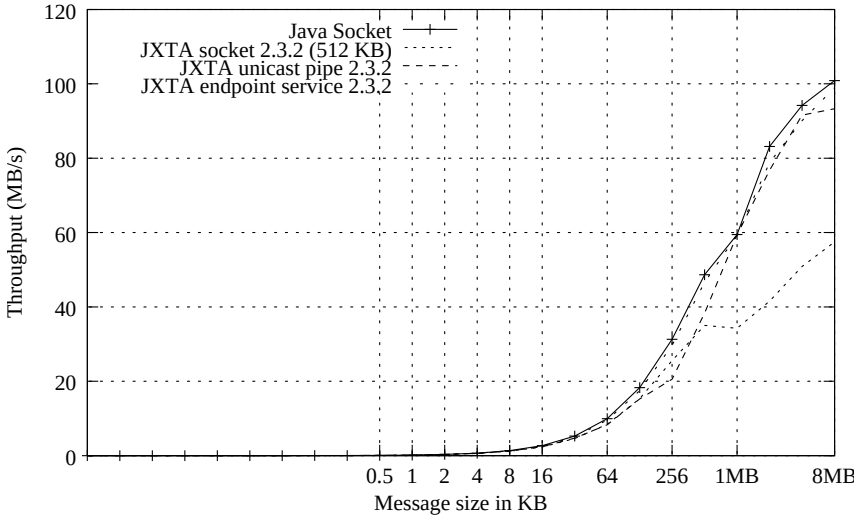


Figure 13: Bandwidth of each layer for JXTA-J2SE 2.3.2 compared to Java sockets over a high-bandwidth WAN.

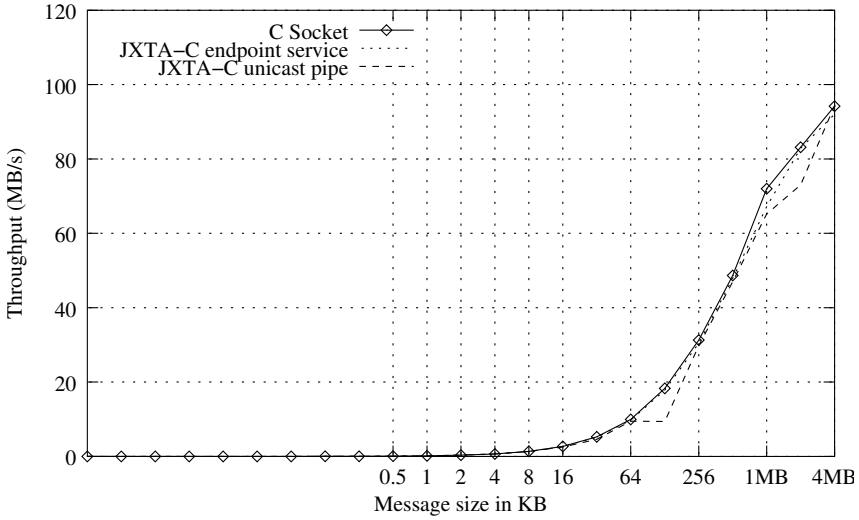


Figure 14: Bandwidth of each layer for JXTA-C 2.1.1 compared to C sockets over a high-bandwidth WAN.

Discussion. Based on this evaluation, we can conclude that JXTA's communication layers, when used on high-bandwidth WANs, are able to reach the same bandwidths as for SAN benchmarks. Both versions of JXTA-J2SE (and not only JXTA-J2SE 2.2.1) as well as JXTA-C 2.1.1 are able to efficiently use the bandwidth available on the links used for interconnecting sites of a grid. However, JXTA sockets are unable to successfully exploit the bandwidth available on WANs, even if JXTA sockets 2.3.2 achieve some improvement compared to previous versions.

8 Discussion

JXTA aims at providing generic blocks for building P2P services or applications. Such services or applications may have various requirements with respect to the performance of inter-peer communications, but also with respect to the desired guarantees. It is, therefore, necessary to pick the appropriate communication layer according to the application requirements, and to properly configure it in order to efficiently use JXTA.

To sum up, this performance evaluation of JXTA communication layers has shown that both bindings of JXTA: 1) can saturate a Fast Ethernet LAN network; 2) can achieve peak bandwidth of around 1 Gb/s over SANs and WANs, at least for the endpoint service and the unicast pipe; 3) exhibits high latency results, especially the unicast pipe and JXTA sockets. The fastest layer of JXTA is clearly the endpoint service. However, direct use of the endpoint service is not recommended, as communications are unreliable and only suitable for static point-to-point interactions. Moreover, this layer may be subject to short-term modifications, which may result in large amounts of work when upgrading to newer versions of JXTA. On the other hand, this layer provides the developer with full control of the logical topology and therefore allows one to implement alternative routing schemes. Therefore, a direct use of this layer is reserved to JXTA experts willing to develop highly specific P2P systems.

As regards the upper layers, using JXTA sockets or JXTA pipes is not an easy choice, at least on SANs and WANs. More specifically, the overhead introduced by the JXTA sockets on LANs compared to the underlying unicast pipes is low, if we take into consideration the features offered by this layer: reliable, bidirectional communications and the availability of a data-stream mode. Note however that this overhead is low only when JXTA sockets are configured to use larger output buffer size. In contrast, on SANs and WANs, the performance degradation of JXTA sockets compared to unicast pipe is high, therefore choosing unicast pipes seems more appropriate.

The good performance of JXTA for Fast Ethernet LANs and high-speed WANs, at least in terms of bandwidth capability, makes JXTA a particularly good candidate for many wide-

area Internet applications dealing with large data transfers over such networks. We can therefore say that JXTA-based collaborative platform such as JXCube [33] or projects supporting distributed computing on large data sets such as P3 [25] or JNGI [30], to name a few, have made a reasonable choice by using JXTA.

Different considerations need to be taken into account when using JXTA as a means of achieving a form of convergence of P2P and grid middleware. In this context, it is an important challenge to allow JXTA-based applications targeting grid infrastructures to transparently exploit these high performance networks. We have shown that, if is correctly tuned, the JXTA platform can deliver adequate performance (e.g., over 1 Gbit/s bandwidth). Furthermore, we explain how this performance can be improved thanks to specialized external libraries. The overall conclusion is that JXTA may provide adequate communication performance that are required by grid computing applications.

However, these evaluations have revealed some weaknesses of JXTA's communication layers: 1) with JXTA-J2SE, the bandwidths of all layers have degraded since JXTA 2.3, hindering JXTA from saturating SAN links; 2) the communication layers of JXTA are not optimized for direct connections between peers therefore limiting the bandwidth and latency results on SANs. Therefore, in spite of our preliminary efforts, both bindings of JXTA still need some improvements in order to be able to fully exploit the available networks provided by grid infrastructures, especially SANs.

By way of comparison with other middlewares, let us cite a few latency results results for various platforms typically used for building grid applications. We use [9] as a reference paper for the latency results of J2SE-based middlewares. However, given that the hardware setup used in this study is not identical to ours, we can only sketch rough trends. In [9], three types of platforms are benchmarked: Object Request Brokers (ORB), component-oriented platforms and web services. We can see that at least the 294 μs latency of JXTA 2.3.2's endpoint service outperforms middlewares in all three categories (517 μs for Java RMI over IIOP [19], 633 μs for OpenORB 1.4.0 [22] and 2070 μs for ProActive 2.0 [17], to name one in each category). Nevertheless, some ORB middlewares and component-oriented platforms, like ORBacus [28] (115 μs) or Java RMI over JRMP [20] (123 μs) and Fractal RMI [21] (151 μs) respectively, achieve better results. Note however that, for a fair comparison, component-oriented platforms and web services should be compared to the higher communication layers of JXTA, such as unicast pipes and JXTA sockets. In that case, JXTA-J2SE layers do not range in the top any longer, even if they still outperform some platforms based on web services (4742 μ for Apache Axis 1.1 [12]). As regards JXTA-C, according to results available with the CORBA benchmark project [31], the endpoint and pipe layers can achieve similar results with OmniORB 3.0 [13] (173 μs) and ORBacus 4.0 (338 μs) respectively. However, they are still far away from the performances of OmniORB

4.0 (52 μs). Overall, we can however conclude JXTA's communication layers performs reasonably well given the provided functionalities.

9 Conclusion

In the context of the current efforts for building grid services on top of P2P libraries, an important question is: to what extent is it reasonable to rely on P2P mechanisms to support the dynamic character of the grid? Are P2P techniques only useful for resource discovery? Or is there a way to take one step further, and efficiently exploit P2P data communication mechanisms? The question of the adequacy of P2P communication mechanisms for performance-constrained usages is therefore important in this context.

In this paper, we focus on benchmarking a key aspect of one widespread P2P open-source library: the performance of JXTA communication layers. We provide a detailed analysis and discussion of the performance of these layers for the most advanced bindings of JXTA (J2SE and C) over different types of networks such as LANs, but also SANs and WANs, typically used for building grid infrastructures. We show that the JXTA platform can deliver adequate performance on grid infrastructures (e.g., over 1 Gbit/s bandwidth) if it is correctly tuned. Moreover, we explain how this performance can be further improved thanks to specialized external libraries. We also give some hints to designers of JXTA-based applications or services on how to efficiently use each layer. This should allow developers to build higher-level services based on building blocks whose costs are known and optimized, which should lead to reasonable implementation choices.

However, these evaluations have revealed some weaknesses of JXTA in both SAN and WAN areas. JXTA-J2SE peak bandwidth values has degraded since JXTA 2.3, hindering JXTA from saturating SAN links. Moreover, the communication layers of JXTA are not optimized for direct connections available on SANs. Therefore, in spite of our initial efforts, JXTA-C still needs some improvements in order to be able to fully and transparently exploit low latency capabilities of SANs through the use of PadicoTM. We plan to achieve this by optimizing message composition emits by JXTA-C when direct communications are available as well as optimizing message path throughout the JXTA-C stack. We believe that this will provide the ability to reach a bandwidth of over 200 MB/s through the use of PadicoTM. For JXTA-J2SE, we hope that PadicoTM will support 1.4 compliant JVMs in a near future. As a step further in improving JXTA's performance on WANs, we plan to introduce the use of parallel streams for both bindings of JXTA, in order to fully exploit the available bandwidth. Again, thanks to PadicoTM, this functionality will be transparently available to JXTA-C (whereas for JXTA-J2SE this would still require some implementation

effort to support this functionality). Finally, it would also be interesting to measure the impact of on-the-fly compression techniques available in PadicoTM for WAN transfers.

Even if this paper explored the impact of multiple factors, this research is not an exhaustive evaluation of all aspects that may impact on JXTA's communication performance. In particular, the work presented in this paper could be extended by using different virtual network topologies, involving more complex communication schemes (e.g. involving communication between peers that are not directly connected). Finally, similar studies for different types of JXTA pipes, such as secure unicast pipes, would also be helpful for JXTA service designers.

References

[1] Michele Amoretti, Gianni Conte, Monica Reggiani, and Francesco Zanichelli. Service Discovery in a Grid-based Peer-to-Peer Architecture. In *International Workshop on e-Business and Model Based IT Systems Design*, Saint Petersburg, Russia, April 2004.

[2] Gabriel Antoniu, Luc Bougé, and Mathieu Jan. JuxMem: Weaving together the P2P and DSM paradigms to enable a Grid Data-sharing Service. *Kluwer Journal of Supercomputing*, 2004. To appear. Preliminary electronic version available at URL `http://www.inria.fr/rrrt/rr-5082.html`.

[3] Gabriel Antoniu, Phil Hatcher, Mathieu Jan, and David A. Noblet. Performance Evaluation of JXTA Communication Layers. In *5th International Workshop on Global and Peer-to-Peer Computing (GP2PC '05)*, Cardiff, UK, May 2005. Held in conjunction with the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '2005).

[4] Gabriel Antoniu, Mathieu Jan, and David A. Noblet. Enabling the P2P JXTA platform for high-performance networking grid infrastructures. In *Proc. of the 2005 Conf. on High Performance Computing and Communications (HPCC '05)*, Napoli, Italy, September 2005. To appear.

[5] Sébastien Baehni, Patrick Th. Eugster, and Rachid Guerraoui. OS Support for P2P Programming: a Case for TPS. In *22nd International Conference on Distributed Computing Systems (ICDCS '02)*, pages 355–362, Vienna, Austria, July 2002. IEEE Computer Society.

[6] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proc. of the 2003 Conf. on Appli-*

cations, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03), pages 407–418, New York, NY, USA, 2003. ACM Press.

[7] Calvin Cheng, Eric Pouyoul, and Mathieu Jan. JXTA Distributed Framework. `http://jdf.jxta.org/`, 2003.

[8] V. Danjean, R. Namyst, and R. Russell. Integrating Kernel Activations in a Multithreaded Runtime System on Linux. In *Parallel and Distributed Processing. Proc. 4th Workshop on Runtime Systems for Parallel Programming (RTSPP '00)*, volume 1800 of *Lect. Notes in Comp. Science*, pages 1160–1167, Cancun, Mexico, May 2000. In conjunction with IPDPS 2000. IEEE TCPP and ACM, Springer-Verlag.

[9] Christophe Demarey, Gael Harbonnier, Romain Rouvoy, and Philippe Merle. Benchmarking the round-trip latency of various java-based middleware platforms. *Studia Informatica Universalis Regular Issue*, 4(1):7–24, May 2005.

[10] Alexandre Denis, Christian Pérez, and Thierry Priol. PadicoTM: An Open Integration Framework for Communication Middleware and Runtimes. *Future Generation Computer Systems*, 19(4):575–585, May 2003.

[11] Ian Foster and Adriana Iamnitchi. On Death, Taxes, and the Convergence on Peer-to-Peer and Grid Computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, number 2735 in Lect. Notes in Comp. Science, Berkeley, CA, February 2003. Springer-Verlag.

[12] The Apache Software Foundation. WebServices - Axis. *http://ws.apache.org/axis/*, 2003.

[13] Duncan Grisby. OmniORB Web Site. *http://omniorb.sourceforge.org*, 2002.

[14] Emir Halepovic and Ralph Deters. The Cost of Using JXTA. In *3rd International Conference on Peer-to-Peer Computing (P2P '03)*, pages 160–167, Linköping, Sweden, September 2003. IEEE Computer Society.

[15] Emir Halepovic and Ralph Deters. JXTA Performance Study. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM '03)*, pages 149–154, Victoria, B.C., Canada, August 2003. IEEE Computer Society.

[16] Emir Halepovic and Ralph Deters. JXTA Messaging: Analysis of Feature-Performance Tradeoffs. Submitted for publication, 2005.

[17] INRIA. ProActive Web Site. *http://proactive.objectweb.org*, 2002.

[18] Markus Junginger and Yugyung Lee. The Multi-Ring Topology - High-Performance Group Communication in Peer-to-Peer Networks. In *2nd International Conference on Peer-to-Peer Computing (P2P '02)*, pages 49–56, Linköping, Sweden, September 2002. IEEE Computer Society.

[19] Sun Microsystems. Java Remote Method Invocation (RMI) over IIOP Documentation. *http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop/*, 2002.

[20] Sun Microsystems. Java Remote Method Invocation Specification. Revision 1.8, Sun Microsystems Inc., 2002. *http://java.sun.com/j2se/1.4.2/docs/guide/rmi/*.

[21] ObjectWeb. The Fractal Project. *http://fractal.objectweb.org*, 2002.

[22] The OpenORB Team. The Community OpenORB Project. *http://openorb.sourceforge.net*, 2002.

[23] Jean-Marc Seigneur. Jxta Pipes Performance. `http://bench.jxta.org/papers/jmjxtapipesperformance.pdf`, 2002.

[24] Jean-Marc Seigneur, Gregory Biegel, and Christian Damsgaard Jensen. P2P with JXTA-Java pipes. In *2nd international Conference on Principles and Practice of Programming in Java (PPPJ '03)*, pages 207–212, Kilkenny City, Ireland, 2003. Computer Science Press, Inc.

[25] Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. P3: Personal Power Plant. GGF10: Open Grid Service Architecture - Peer-to-Peer Research Group (OGSA-P2P RG), March 2004.

[26] Domenico Talia and Paolo Trunfio. Toward a Synergy Between P2P and Grids. *IEEE Internet Computing*, 7(4):94–96, 2003.

[27] Domenico Talia and Paolo Trunfio. A P2P Grid Services-Based Protocol: Design and Evaluation. In *Euro-Par 2004: Parallel Processing*, number 3149 in Lect. Notes in Comp. Science, pages 1022–1031, Pisa, Italy, August 2004. Springer-Verlag.

[28] IONA Technologies. ORBacus Web Site. *http://www.orbacus.com*, 2004.

[29] Phong Tran, Jeffrey Gosper, and Albert Yu. JXTA and TIBCO Rendezvous - An Architectural and Performance Comparison. `http://www.smartspaces.csiro.au/docs/PhongGosperYu2003.pdf`, 2003.

[30] Jerome Verbeke, Neelakanth Nadgir, Greg Ruetsch, and Ilya Sharapov. Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. In *3rd International Workshop on Grid Computing*, Lect. Notes in Comp. Science, pages 1–12, Baltimore, MD, November 2002. Springer-Verlag.

[31] Open CORBA Benchmarking. `http://nenya.ms.mff.cuni.cz/~bench/`.

[32] Grid'5000 project. `http://www.grid5000.org/`, 2004.

[33] JXCube - Jxta eXtreme Cube project. `http://jxcube.jxta.org/`.

[34] The JXTA project. `http://www.jxta.org/`, 2001.

[35] Cog Kit JXTA project. `http://www-unix.globus.org/cog/projects/jxta/`.

[36] JXTA-Grid project. `http://jxta-grid.jxta.org/`.

[37] JXTA specification project. `http://spec.jxta.org/`.

[38] KaZaA. `http://www.kazaa.com/`.

[39] Project jxta-c. `http://jxta-c.jxta.org/`.