Performance Evaluation of JXTA Communication Layers

Gabriel Antoniu, Phil Hatcher, Mathieu Jan, David A. Noblet

▶ To cite this version:

HAL Id: inria-00000973

https://inria.hal.science/inria-00000973

Submitted on 9 Jan 2006

Performance Evaluation of
JXTA Communication Layers

Gabriel Antoniu*, Phil Hatcher†, Mathieu Jan* and David A. Noblet†

*IRISA/INRIA, Campus de Beaulieu 35042 Rennes Cedex, France

†University of New Hampshire Department of Computer Science Durham, New Hampshire 03824-3591, U.S.A

Email: Mathieu.Jan@irisa.fr

Abstract— The arrival of the P2P model has opened many new avenues for research within the field of distributed computing. This is mainly due to important practical features (such as support for volatility, high scalability). Several generic P2P libraries have been proposed for building higher-level services. In order to judge the appropriateness of using a generic P2P library for a given application type, an *experimental* performance evaluation of the provided functionalities is unavoidable. Very few analyses of this kind have been reported, as most evaluations are limited to complexity analyses and to simulations. Such experimental analyses are important, especially when using P2P software in a grid computing context, where applications may have precise efficiency requirements. In this paper, we focus on JXTA, which provides generic building blocks and protocols intended to serve as a basis for specialized P2P services and applications. We perform a performance evaluation of the three communication layers (endpoint, pipe and socket) over a Fast Ethernet local-area network, for recent versions of the J2SE and C bindings of JXTA. We provide a detailed analysis explaining the behavior of these three layers and we give hints showing how to efficiently use them.

I. INTRODUCTION

The pioneering work resulting from the development of Peer-to-Peer (P2P) systems, such as Gnutella, has highlighted many interesting properties of P2P, such as high scalability and high availability of service despite highly dynamic changes in the underlying physical infrastructure. Thanks to these desirable features, the P2P interaction model has been very successful and has recently been an influential player in many research communities. Therefore, a shift to the P2P model has become attractive for many classes of applications originally based on the traditional client-server model (e.g. collaborative applications, instant messaging, etc.). Furthermore, a growing number of projects have been quick to embrace the P2P model directly from their initial design phase.

Recently, a number of P2P libraries (e.g. FreePastry [1], JXTA [2], etc.) providing basic support for P2P interaction (for example discovery mechanisms) have been made available to the research community. Such libraries are intended to serve as generic building blocks for higher-level P2P services and applications.

However, before using such generic layers, it is important to analyze their suitability with respect to the requirements of the target P2P service or application. Most published papers introducing these libraries give a detailed overview of their design, but generally omit the necessary detailed experimental evaluations that would allow potential users to understand the behavior of the system in practice. For instance, the P2P algorithm community has mainly focused its research activity on the development of overlay networks based on DHTs, where communication cost is modeled as a distance expressed in the number of logical hops. The cost of the basic operations (e.g. routing and discovery) is evaluated through complexity analyses and simulations. This kind of theoretical evaluation is certainly necessary, but it is clearly only a preliminary step. To fully understand the behavior of the proposed P2P libraries, *experimental evaluations* on existing distributed testbeds are unavoidable. Such practical evaluations are able to capture aspects related, for instance, to *physical locality* or *specifics of the underlying physical networks*, often ignored by theoretical evaluations.

This work focuses on the performance of a particular P2P library, namely JXTA. The choice of concentrating on the JXTA project is motivated by the fact that, to the best of our knowledge, it is the most advanced framework currently available for building services and applications based on the P2P model. JXTA is an open-source initiative, sparked by Sun Microsystems, founded in order to develop a set of standard protocols designed to support P2P network applications. In its 2.0 version, JXTA consists of a specification of six language- and platform-independent, XML-based protocols [3] that provide basic services common to most P2P applications, such as peer group organization, resource discovery, and inter-peer communication. A more detailed overview of JXTA can be found in [4]. These generic protocols require some specialization, however, in order to match specific application requirements. Therefore, obtaining a clear picture with respect to the performance characteristics of JXTA is necessary before attempting to use it in the development of any specific P2P services. For example, have JXTA-based collaborative platform such as JXCube [5] or projects supporting distributed computing on large data sets such as P3 [6] or JNGI [7], to name a few, made a reasonable choice when using JXTA? In this paper, we focus on the evaluation of one important aspect of JXTA: the performance of its three communication layers (endpoint layer, pipe layer and socket layer). The most complete bindings of the JXTA protocols are the J2SE reference implementation (denoted JXTA-J2SE) and the recently updated C implementation [8] (denoted JXTA-C). Other bindings exist, however not enough development has been

done on them yet to produce meaningful results.

In order to evaluate the cost of JXTA communications, we perform a number of *bidirectional bandwidth tests* (also known as ping-pong tests) between JXTA peers. We perform these tests over a Fast Ethernet local-area network for both JXTA-J2SE and JXTA-C, using each of the available JXTA communication layers (varying certain experimental parameters such as message size, buffer size and JVM options). Although such a basic benchmark cannot provide a comprehensive evaluation of the performance of a P2P library, as direct communication between peers may be the exception rather than the rule (because of firewalls, etc.) and also because P2P systems are generally deployed on wide-area networks, it still highlights the inherent strengths and weaknesses of the performance of direct inter-peer communication. In addition, we use measurements of *protocol efficiency* that we obtain for each communication layer to help us analyze the results obtained in the bidirectional bandwidth tests.

The remainder of the paper is organized as follows. Section II introduces related work: we discuss some existing performance evaluations for older versions of JXTA-J2SE. Section III provides an overview of the communication layers of both JXTA-J2SE and JXTA-C. Section IV describes the experimental setup used for the benchmarks. Sections V and VI present the results obtained from performing the specified benchmarks and give a corresponding analysis of the cost of each layer for JXTA-J2SE and JXTA-C, respectively. In Section VII we discuss the measurements from a global perspective and provide some hints as to how to make efficient use of the JXTA communication layers. Finally, Section VIII concludes the paper and suggests directions for additional research.

II. RELATED WORK

In this paper, we focus on the performance of the J2SE and C bindings of JXTA. Let us note, however, that no results about the performance of JXTA-C have been published so far. Consequently, any reference to "JXTA" in this section will refer to the J2SE binding.

The performance of JXTA has been compared a number of times to various other P2P systems [9–11]. These studies highlight the high overhead introduced by JXTA and the difficulty encountered in the use of the library. However, [10] and [11] compare themselves to old, unoptimized versions of JXTA (prior to 1.0). An attempt to use JXTA to build near-real-time applications has led to the conclusion that the default XML parser used inside JXTA has poor performance [12], therefore making JXTA unsuitable for building time-constrained applications. This study is based on JXTA 2.1, however the paper shows that JXTA can be configured to use other XML parsers, providing a sharp increase in its performance. The introduction of a loosely-consistent DHT in JXTA 2.0 [13] has also been the subject of a study [14]. This study compares the approach taken by JXTA to a centralized or flooding approach (which was the strategy of JXTA 1.0), with respect to query response time (for different configurations of a JXTA virtual network),

memory usage and reliability. However, no comparison with other existing DHTs is reported.

To narrow the subject even more, in our work we only concentrate on the communication layer of JXTA-J2SE and JXTA-C. Most published papers on this topic have focused on the widely-used communication layer of JXTA-J2SE: the pipe service. However, these studies are primarily based on JXTA 1.0 [15], [16] or even older [17], [18]. Newer versions, starting with JXTA 2.0, have been claimed to introduce significant design enhancements making these results obsolete. To our knowledge, only two papers have published results about JXTA 2.0 ([16], [19]). Additionally, in all studies benchmarks are performed at the application-level without an in-depth analysis of the results at the level of the communication layers. In [15], the authors define a performance model for JXTA based on the analysis of typical peer operations, along with pipe message round-trip time (RTT), pipe message and data throughput metrics. The closest related work is [19] since tests were performed not only for the pipe service but also for JXTA sockets. However, these benchmarks seem to have been conducted for JXTA 2.2 (sometimes 2.0 is stated), 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 are proposed. In other studies, some performance results with respect to latency performance are inconsistent (e.g. [6] on one side and [16], [19] on the other side). This makes it hard to get a clear view of the performance of JXTA communication layers.

Finally, one particular project worth noting with respect to JXTA performance evaluation is the JXTA Bench project [20], whose goal is to collect and report information about the different aspects of JXTA performance. The project site proposes a plan for the benchmarking of JXTA and the integration of tests into the project. However, few of these benchmarks have been performed and no in-depth analysis of the available results has been published so far.

III. 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, of course, 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*
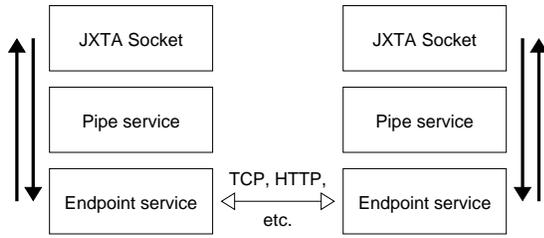
Fig. 1. Stack of JXTA communication protocols.

elements [3], 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.

A. 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, all communications at the endpoint level, regardless of the underlying transport protocol, are asynchronous, unidirectional and unreliable.

Using the interface that the endpoint service provides, all the information that one peer must have 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 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 service in charge of handling the received message.

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.

B. Core communication layer: the pipe service

The pipe service supplements the endpoint service by incorporating the abstraction of virtual communication channels (or *pipes*). 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). Therefore, the pipe service provides the illusion of a virtual endpoint that is independent of any single peer location and network topology, as stipulated by JXTA specifications.

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. And, 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 the basis for the implementation of the higher-level JXTA sockets.

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, another 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 even when a direct connection is available between peers.

C. 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 unpackaging 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 is not implemented in JXTA-C. It was introduced in JXTA-J2SE 2.0, with reliability support added in 2.1.

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, the value that influences how 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.

IV. PRACTICAL DETAILS ABOUT THE EXPERIMENT

The bidirectional bandwidth test was chosen because it is a very basic performance metric frequently used to benchmark many other networking protocols, and because of its ability to yield information about important performance characteristics such as bandwidth and latency. Basically, this benchmark consists of a back-and-forth exchange of an identical message between two peers. Each test is comprised of successive measurements taken over a range of varying message payload sizes, from 1 byte up to 16 MB. All measurements are sampled at the application level and are calculated based on five subsequent time measurements of the exchange of 100 consecutive message-acknowledgment pairs. When JXTA-J2SE is benchmarked, an additional warm-up period of 1,000 message-acknowledgments is performed, to make sure that the Just-In-Time (JIT) compiler is not disturbing the measurements. It should be noted that all source code required to perform the benchmarking of each communication layer of JXTA-J2SE has been made available via the web site of project JDF [21].

Protocol efficiency is the other factor explored in the performance evaluation of the JXTA protocols. Protocol efficiency 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.

The nodes used for these benchmarks 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; the hardware network layer used is a Fast Ethernet (100 Mb/s) local-area network. Tests were executed using JXTA-J2SE 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 TCP Nagle's algorithm). Both bindings were configured to use TCP as the underlying transport protocol. When tests are performed using JXTA-J2SE, the
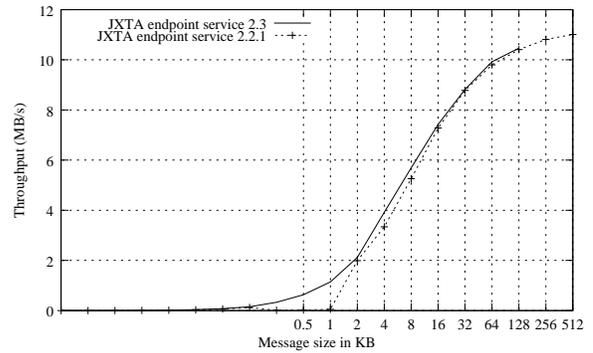

Fig. 2.   Endpoint service throughput using JXTA 2.2.1 and 2.3.

Sun Microsystems Java Virtual Machine (JVM) 1.4.2_01-b06 is used as the default JVM; the JVM is executed with `-server -Xms256M -Xmx256M` options. The JVM 1.4.1 from IBM is also used at times with the following options: `-Xms256M -Xmx256M`; such uses of the IBM JVM are explicitly stated in the performance analysis. It should be noted that, because of the use of `javax` classes in a required library of JXTA, JXTA 2.2.1 does not run on top of the IBM JVM. Finally, the JXTA-C benchmarks are compiled using `gcc` 3.3.3 with the `O2` level of optimization.

V. PERFORMANCE EVALUATION OF JXTA-J2SE

This section presents an analysis of the results obtained for the performance of JXTA-J2SE communication layers running over a Fast Ethernet local-area network. Each subsection presents the performance of one communication layer, the last one giving an overall view and making a comparison with Java sockets.

A. JXTA-J2SE Endpoint Service

Figure 2 shows the bandwidth curves for the endpoint service using JXTA 2.2.1 and 2.3. For the most part, this figure highlights the similarities between these two versions, although it does show that the latter achieves slightly better results up to the limit imposed on the size of messages for this version of JXTA. 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, it was lowered from 512 KB to 128 KB (of application-level payload). Therefore, it is not surprising that JXTA 2.2.1 achieves a higher peak throughput (11.01 MB/s) as compared to its 2.3 counterpart (10.47 MB/s). By modifying the source code of JXTA we removed this limit, allowing large messages to be transmitted. Consequently, JXTA 2.2.1 and 2.3 attain peak throughputs of 11.20 MB/s and 11.15 MB/s, respectively, during the transmission of a 4 MB message. This is an increase of 6% for JXTA 2.3 as compared to the throughput exhibited by the default endpoint service limited to message sizes of 128 KB.
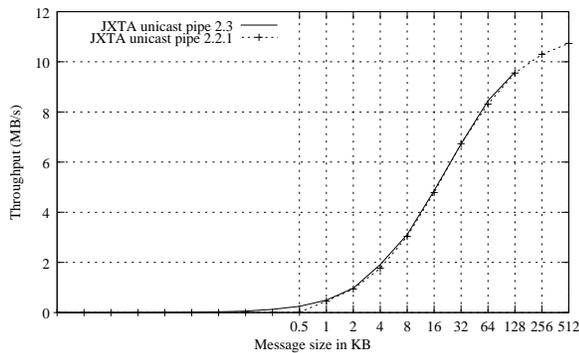
Fig. 3.   Unicast pipe throughput using JXTA 2.2.1 and 2.3.

Fig. 4.   JXTA socket throughput using JXTA 2.2.1 and 2.3.

On the latency side, the endpoint service for both versions of JXTA achieve latency measurements in the sub-millisecond range: 960 usec for JXTA 2.2.1 and 735 usec for JXTA 2.3; with the IBM JVM, JXTA 2.3 even achieves a latency of 485 usec. Additionally, the latency is also affected by transmitting two TCP packets for every JXTA message (issue 1228 which has since been fixed in JXTA 2.3.1). Still, when using the IBM JVM, the latency result only improves by 16 usec. Note that the protocol efficiency of JXTA's lowest layer is: 300 bytes for a 1-byte message payload.

B. JXTA-J2SE Unicast Pipe

As for the endpoint layer, Figure 3 illustrates the similarity between the shapes of the JXTA unicast pipe versions 2.2.1 and 2.3. Again, as with the endpoint service, the message size limit prevents JXTA 2.3 throughput (9.59 MB/s) from reaching the higher throughput of JXTA 2.2.1 (10.74 MB/s). By removing this limit, the peak throughput of JXTA 2.2.1 increases to 11.14 MB/s. This is a significant improvement (14%) over the peak throughput 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. JXTA 2.3 yields latency results 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 result for JXTA 2.3, as compared to the endpoint layer, is explained by the additional message element added by the pipe service into the JXTA message. As described in Section III, the pipe 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
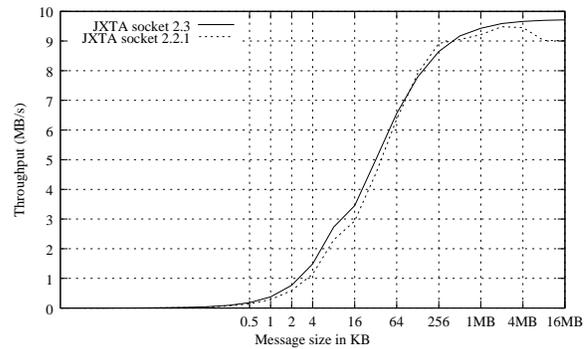
protocol efficiency decreases to less than half that of the endpoint layer).

C. JXTA-J2SE Sockets

Figure 4 shows that *in their default configurations* JXTA sockets 2.2.1 and 2.3 reach maximum throughputs of 9.48 MB/s and 9.72 MB/s, respectively. The low throughputs compared to plain sockets (around 11.22 MB/s in our tests) 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. However, as described in Section III, JXTA can be configured with different values for this parameter. Figure 5, consequently, shows the bandwidth results we obtained for JXTA sockets 2.2.1 and 2.3 using a set of increasing output buffer sizes. For both versions, increased output buffer sizes noticeably increase the throughput for the larger message sizes. With a buffer of 512 KB, JXTA sockets 2.2.1 achieve a peak throughput of 11.12 MB/s and, with a buffer of 128 KB, JXTA sockets 2.3 reach 10.96 MB/s. This is a 17.3% and 12.75% increase in the peak throughputs compared to the performance obtained by the default buffer size for JXTA 2.2.1 and JXTA 2.3, respectively. Also note the small decrease in performance for each of the different buffer sizes at the point in the curve where the JXTA socket has to begin to split up messages.

On the latency side, JXTA sockets express results of around 3.4 ms for version 2.2.1 and 2.5 ms for 2.3. 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 unicast pipes: for a 1-byte message payload, the total size of the JXTA message that is actually transferred is 913 bytes.
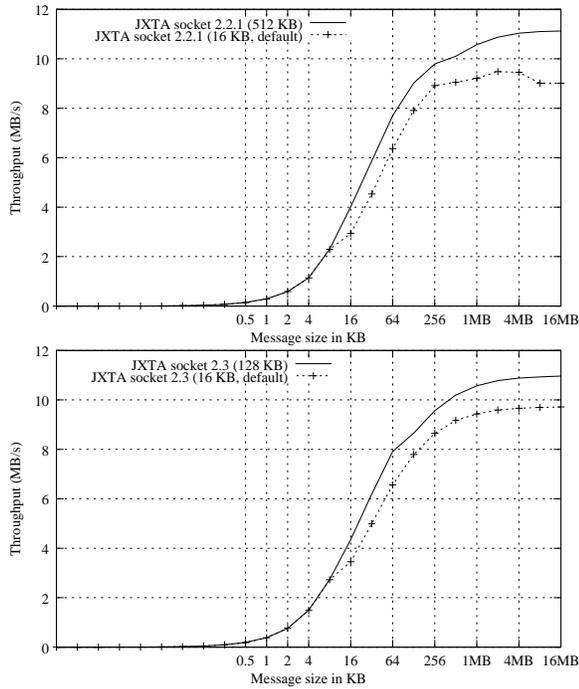
Fig. 5. JXTA socket bandwidth at varying output buffer sizes using JXTA 2.2.1 and 2.3.
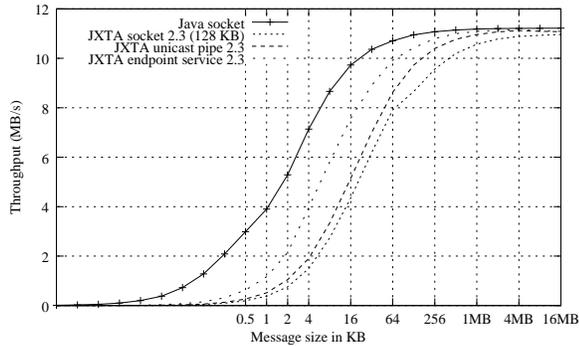

Fig. 6. Throughput of each layer for JXTA 2.3 compared to Java sockets.

D. The Big Picture of JXTA-J2SE Communications Performance

In order to gain a useful perspective on the results obtained, this section juxtaposes the performance of each communication layer of JXTA-J2SE 2.3 with the results obtained using Java sockets (this discussion can similarly be applied to JXTA 2.2.1). For a fair comparison, results are reported based on a modified version of JXTA, where the limit on message size was removed; JXTA sockets are configured to transmit pipe messages of 128 KB.

Figure 6 shows that the performance difference between JXTA sockets and JXTA pipes for sending large messages is negligible. More generally, the curves show that the two main JXTA transport mechanisms directly used by JXTA-based applications are both able to reach the throughput of plain sockets on a Fast Ethernet local-area network.

Java socket	< 0.10 ms
Endpoint service	0.48 ms
Unicast pipe	1.22 ms
JXTA socket	1.76 ms

TABLE I

LATENCY RESULTS FOR JXTA 2.3 COMPARED WITH JAVA SOCKETS.
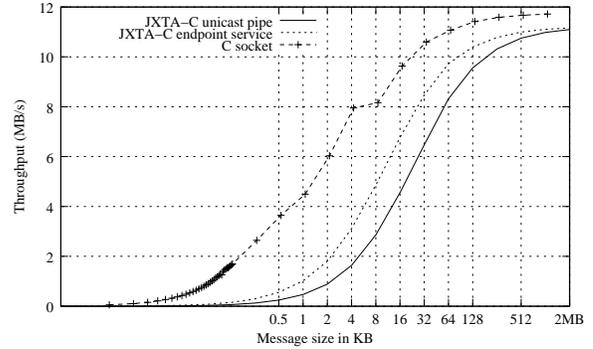

Fig. 7. Throughput of each layer for JXTA-C compared to C sockets.

Table I 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 usec higher. This overhead has not been explained so far, however, we suspect thread related problems, such as scheduling or creation/destruction of threads, as tests have demonstrated the use of approximately 35 threads. The actual number of threads may vary between 33 and 40. It is interesting to note that latency improvements have been observed when using the 2.6 version Linux kernel and its new thread library.

VI. PERFORMANCE EVALUATION OF JXTA-C

Similarly to the previous section, a performance evaluation of JXTA-C communication layers over a Fast Ethernet local-area network is reported in this section. However, note that the JXTA socket layer is not implemented in JXTA-C. Moreover, JXTA-C is a reviving project, developed by fewer people and consequently struggling to reach the JXTA-J2SE level of features. Therefore, this analysis is somewhat shorter than its J2SE counterpart.

A. JXTA-C Endpoint Service

Figure 7 shows the bandwidth and latency measurements of the endpoint service for JXTA-C. The peak throughput of the endpoint service is 11.16 MB/s compared to 11.7 MB/s for plain sockets. Note that no limit on the message size is

C socket	< 0.10 ms
Endpoint service	0.82 ms
Unicast pipe	1.99 ms

TABLE II

LATENCY RESULTS FOR JXTA-C COMPARED WITH C SOCKET.

currently implemented in JXTA-C. The latency measurements of JXTA-C, around 820 usec, is still much higher than the 74 usec latency of plain sockets, as shown in Table II. The difference in the latency between the J2SE and C bindings of JXTA is mainly due to the lack of buffering in the endpoint layer. As we disabled the TCP packet aggregation algorithm to reduce latency, an explicit buffering scheme in the endpoint layer is required so that one TCP packet is sent for a single 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. The protocol efficiency of the endpoint service is slightly better than its J2SE counterpart: 239 bytes bytes for 1-byte of application payload. This difference of efficiency is mainly explained by unspecified encoding tags for each message element in JXTA-C, when default ones are used, compared with JXTA-J2SE which specifies them all the time.

B. JXTA-C Pipe Service

Figure 7 also shows that the peak throughput of unicast pipes is 11.1 MB/s compared to 11.7 MB/s for plain sockets. As illustrated in Table II, latency results for JXTA-C are around 2 ms, much higher than plain sockets and much higher than the latency of 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 of JXTA-C is slightly better than JXTA-J2SE: for a 1-byte message payload, the total size of the JXTA message that is actually transferred is 834 bytes.

VII. DISCUSSION

Bidirectional bandwidth benchmarks show that each communication layer of both JXTA-J2SE and JXTA-C is able to reach the throughput of plain sockets on a Fast Ethernet local-area network. However, JXTA exhibits high latency values as compared to plain sockets. For instance, the widely-used JXTA-J2SE unicast pipes are not able to achieve latencies in the sub-millisecond range. This is mainly due to the presence of a large XML message element in each pipe message (which takes up more than 60% of the total message size for a 1 byte payload). The presence of this element requires costly XML parsing, which is useless when direct connectivity exists between the communicating peers. In such a case, this element is not used by the endpoint router protocol and could be removed from messages in all communication layers. Improvement in this area is expected [22]. The same optimization should also improve the latency results of JXTA-C. Note that the performance of JXTA-C could further be improved through better buffer handling. A zero-copy strategy, as available in JXTA-J2SE, would clearly help the C binding approach the performance of the J2SE binding.

JXTA aims to provide 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 configure it in order to efficiently use JXTA. On the bandwidth side, all communication layers achieve the same performance overall, but on the latency side the endpoint service is a clear winner. 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 require 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 for JXTA experts willing to develop highly specific P2P systems. Finally, with respect to JXTA sockets and JXTA pipe service, choosing the former is the obvious choice. Indeed, the overhead introduced by the JXTA sockets is low, given the features offered by this layer: reliable, bidirectional communications and the availability of a data-stream mode. (Besides, using this layer is recommended by the JXTA team of Sun Microsystems). Note however that this overhead is low only when JXTA sockets are configured to use larger output buffer size. On Fast Ethernet local-area networks, the default value of 16 KB is clearly a bad choice for sending large message sizes; improvements of over 10% can be obtained with higher buffer sizes. Note also that the socket layer is not implemented in JXTA-C.

The suitability of JXTA for Fast-Ethernet local-area networks, at least in terms of throughput capability, makes JXTA a particularly good candidate for many applications running on slower-speed networks (i.e. many wide-area internet applications) and dealing with large data transfers over such networks. We can therefore answear to the question of the introduction: JXTA-based collaborative platform such as JXCube or projects supporting distributed computing on large data sets such as P3 or JNGI, to name a few, have made a reasonable choice when using JXTA.

Another class of distributed systems subject to significant research efforts are *grid computing platforms*. A grid aggregates various resources such as storage space, processors, or sensors, in order to provide a global view of these resources (generally made available by multiple institutions). One criticism about currently deployed grids is their lack of flexibility, especially for discovery algorithms. Using routing algorithms based on the P2P approaches is one important hurdle to overcome in the context of the convergence of P2P libraries, such as JXTA, and grid computing middleware [23]. This is, for instance, the goal of a new project called Service-oriented Peer-to-Peer Architecture [24] (SP2A) based on two specifications: the Open Grid Service Infrastructure (OGSI) and JXTA. Within the same context, another important aspect in enabling JXTA for grids regards the efficient use of high bandwidth networks, such as Giga Ethernet or Myrinet, that may be available in the clusters that compose the grid. Therefore, an

important challenge will be to allow JXTA-based applications targeting grid infrastructures to transparently exploit these high performance networks. In such a context, performance evaluation to allow the correct tuning of the JXTA communication layers becomes a necessity.

VIII. CONCLUSION

The promising properties of the P2P model have motivated many projects, both in the academic and industrial world, to adopt this communication model. However, this quick shift has happened in the absence of experimental performance studies indicating the suitability of this model for the target applications.

In this paper, we focus on benchmarking a key aspect of one widespread P2P open-source library: 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 a Fast Ethernet local-area network. Finally, we also give some hints to designers of JXTA-based applications or services on how to efficiently use each layer. This allows developers to build higher-level services based on building blocks whose costs are known and optimized, which should lead to reasonable choices.

Still, in spite of all the factors explored in this paper, this research is not an exhaustive evaluation of all aspects of JXTA communication performance. In particular, tests over different kinds of networks may confirm that the bandwidth of Fast Ethernet networks are a bottleneck for the performance of JXTA. We are currently benchmarking JXTA over high-speed networks, namely Giga Ethernet and Myrinet. Preliminary results show that JXTA is able to achieve throughput above 1 Gb/s. We also have successfully ported JXTA-C to PadicoTM [25], a high-performance framework for networking and multi-threading. However, JXTA-C communication layers require some improvements in order to efficiently use this middleware, especially on the latency side. In addition, we have started to run our benchmarks on wide-area networks in order to verify that the conclusions of this paper still apply on these kind of networks. These experiments are presented and analyzed in [26]. Furthermore, the work presented in this paper could be extended with an evaluation of JXTA communication layers over different virtual network topologies, involving more complex communication schemes (e.g. involving communication between peers that are not directly connected). Similar studies for different types of JXTA pipes (other than unicast pipes) would also be helpful for JXTA service designers. Finally, to aid in performance analysis, it would be helpful to write a plug-in for the popular ethereal network protocol analysis software, in order to make it JXTA-aware.

REFERENCES

[1] "FreePastry," http://freepastry.rice.edu/.
[2] "The JXTA project," http://www.jxta.org/.
[3] "JXTA specification project," http://spec.jxta.org/.
[4] B. Traversat, M. Abdelaziz, M. Duigou, J.-C. Hugly, E. Pouyoul, and B. Yeager, "Project JXTA Virtual Network," http://www.jxta.org/project/www/docs/JXTAprotocols_01nov02.pdf, Oct. 2002.
[5] "JXCube - Jxta eXtreme Cube project," http://jxcube.jxta.org/.
[6] K. Shudo, Y. Tanaka, and S. Sekiguchi, "P3: Personal Power Plant," GGF10: Open Grid Service Architecture - Peer-to-Peer Research Group (OGSA-P2P RG), Mar. 2004.
[7] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov, "Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment," in *3rd International Workshop on Grid Computing*, ser. Lect. Notes in Comp. Science. Baltimore, MD: Springer-Verlag, Nov. 2002, pp. 1–12.
[8] B. Traversat, M. Abdelaziz, D. Doolin, M. Duigou, J.-C. Hugly, and E. Pouyoul, "Project JXTA-C: Enabling a Web of Things," in *36th Annual Hawaii International Conference on System Sciences (HICSS '03)*. Big Island, Hawaii: IEEE Computer Society, Jan. 2003, p. 282b.
[9] S. Baehni, P. T. Eugster, and R. Guerraoui, "OS Support for P2P Programming: a Case for TPS," in *22nd International Conference on Distributed Computing Systems (ICDCS '02)*. Vienna, Austria: IEEE Computer Society, July 2002, pp. 355–362.
[10] M. Junginger and Y. 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)*. Linköping, Sweden: IEEE Computer Society, Sept. 2002, pp. 49–56.
[11] P. Tran, J. Gosper, and A. Yu, "JXTA and TIBCO Rendezvous - An Architectural and Performance Comparison," http://www.smartspaces.csiro.au/docs/PhongGosperYu2003.pdf.
[12] D. C. Parker, S. A. Collins, and D. C. Cleary, "Building Near Real-Time P-2-P Applications with JXTA," in *4th International Scientific Workshop on Global and Peer-to-Peer Computing (GP2PC '04)*. Chicago, USA: IEEE Computer Society, Apr. 2004, held in conjunction with CCGRID 2004.
[13] B. Traversat, M. Abdelaziz, and E. Pouyoul, "Project JXTA: A Loosely-Consistent DHT Rendezvous Walker," http://www.jxta.org/docs/jxta-dht.pdf, Mar. 2003.
[14] E. Halepovic, R. Deters, and B. Traversat, "Performance Evaluation of JXTA Rendezvous," in *International Symposium on Distributed Objects and Applications (DOA '04)*. Agia Napa, Cyprus: Springer Verlag, Oct. 2004, pp. 1125–1142.
[15] E. Halepovic and R. Deters, "The Cost of Using JXTA," in *3rd International Conference on Peer-to-Peer Computing (P2P '03)*. Linköping, Sweden: IEEE Computer Society, Sept. 2003, pp. 160–167.
[16] E. Halepovic and R. Deters, "JXTA Performance Study," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM '03)*. Victoria, B.C., Canada: IEEE Computer Society, Aug. 2003, pp. 149–154.
[17] J.-M. Seigneur, "Jxta Pipes Performance," http://bench.jxta.org/papers/jmjxtapipesperformance.pdf, 2002.
[18] J.-M. Seigneur, G. Biegel, and C. D. Jensen, "P2P with JXTA-Java pipes," in *2nd international Conference on Principles and Practice of Programming in Java (PPPJ '03)*. Kilkenny City, Ireland: Computer Science Press, Inc., 2003, pp. 207–212.
[19] E. Halepovic and R. Deters, "JXTA Messaging: Analysis of Feature-Performance Tradeoffs," http://bosna.usask.ca/pub/JXTAMessagingPerf-toReview.pdf, submitted for publication.
[20] "The JXTA bench project," http://bench.jxta.org/.
[21] "The JXTA Distributed Framework project," http://jdf.jxta.org/.
[22] "JXTA issue 208," http://platform.jxta.org/issues/show_bug.cgi?id=208.
[23] I. Foster and A. 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)*, ser. Lect. Notes in Comp. Science, no. 2735. Berkeley, CA: Springer-Verlag, Feb. 2003.
[24] M. Amoretti, G. Conte, M. Reggiani, and F. 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, Apr. 2004.
[25] A. Denis, C. Pérez, and T. Priol, "PadicoTM: An Open Integration Framework for Communication Middleware and Runtimes," in *IEEE International Symposium on Cluster Computing and the Grid (CC-Grid '02)*. Berlin, Germany: IEEE Computer Society, May 2002, pp. 144–151.
[26] G. Antoniu, M. Jan, and D. A. Noblet, "Enabling jxta for high performance grid computing," INRIA, IRISA, Rennes, France, Research Report RR-5488, Feb. 2005, submitted to the Euro-Par 2005: Parallel Processing.