Performance Analysis of JXTA/JXME Applications in Hybrid Fixed/Mobile Environments

Tomás Piedrahita Edwin Montoya*

Abstract

P2P applications and mobile computing have merged. Mobile devices are now powerful enough to efectively manage important amounts of information that may be of interest to others. Meanwhile, P2P systems have demonstrated their efectiveness in sharing information. One of the leading technologies that have resulted is JXME, the mobile version of Sun Microsystem's JXTA. Although some JXTA benchmark analysis have been performed, these tests on JXME have yet to be realized. This paper proposes a testing model which has been developed and executed to be able to measure certain aspects of the JXME system. The testing model consists of three parts: 1) the typical JXME peer operation's latency, 2) the pipe message round trip time and 3) the pipe message and data throughput. The typical JXME peer operations latency is designed to measure how long it takes to run every one of the operations considered to be executed by a typical peer. The pipe message round trip time measures the time it takes for a minimum size message to travel to another peer and back again. Finally, the pipe message and data throughput is to measure the maximum speed at which data can be sent to another peer. To be able to clearly identify the features that need improvement every one of these tests where executed in JXTA as well. Additionally, due to the device and network constraints of the tests in mobile computing environments, a baseline test was developed to account for different network conditions. This way, developers interested in using JXME can know what to expect and the JXME community can identify what features should be improved.

Keywords: *JXTA, JXME, Benchmark, P2P, Mobile Applications.*

1 Introduction

P2P systems are, in their purest form, descentralized systems. It is the contrary from common client-server systems. Client-server architectures are those where there is one computer or element in the network that contains all the information and services, while the others are simply consuming these services. In P2P systems each computer or element in the network is a client and a server at the same time. In recent years these architectures have had a great impact with systems like SETI@Home and specially with file-sharing systems like Napster and KaZaA.

With the growing amount of P2P systems, Sun Microsystems has noticed that building them is complex and that developers have to solve the same problems over and over again, making the construction and design of these systems very inefficient [10]. To ease these inconveniences, Sun decided to create a middleware, consisting of six protocols that would provide solutions to many of the common problems that developers have to face, and JXTA [1, 2, 8, 9, 10] was born. JXTA is a framework which defines the specification of six protocols. The messages are XML making it independent of language, operating system and network topology [2, 8, 10]. Initially the platform was developed in Java, but soon was ported to many other languages. JXME [3,6] is the version of

* EAFIT University, Carrera 49 –7 Sur 50, Medellín , Colombia {tpiedrah, emontoya}@eafit.edu.co

JXTA that runs on mobile computing environments which use J2ME (Java 2 Micro Edition). Portable devices such as cellular phones and PDAs, different wireless commmunications protocols like GPRS/EDGE, Wi-Fi or Bluetooth. This way, JXME peers can join JXTA networks.

As any new technology, there was a need to measure its performance. So, the "bench" project [4] was created for this. Here, the JXTA Community created a benchmarking plan [5] to be able to measure the performance of the different features the platform offers. Based on this project, a more intuitive and extended performance model was developed by Halvepovic and Deters [1].

Results from these tests have been used to better understand and improve JXTA. However, JXME's performance has not been measured. The idea behind this paper is to build a suite of applications that would execute a set of tests designed specifically for JXME based on the proposed plans from the "bench" project and Halepovic and Deters' work [1]. The idea is to be able to measure the performance of applications developed using these technologies in different environments using PDAs, cellular phones and PCs connected through different networks.

This paper is organized as follows: Section 2 explains the three different tests that were developed; Section 3 describes how the tests were executed and the conditions that they were under; Section 4 shows the results and their analysis and Section 5 proposes future work in this area.

2 JXME

One of the most active proyects in the JXTA Community is JXME (or JXTA-J2ME). This proyect's objective is to enable JXTA compatible functionalities on small devices using J2ME. This allows devices like cellular phones and PDAs to actively participate in P2P activities with other devices inside a JXTA network. However, J2ME has two main versions CDC and CLDC. CDC is intended for devices with limited capacities, but still quite powerful, such as PDAs and CLDC is intended for much more limited devices like cellular phones and sensor networks. Therefore JXME has also two versions: proxied and proxyless. The proxyless version is still under development and not considered in any way in this proyect. This is because it is still unavailable to be executed in real devices because it requires CDC version 1.1 to be released, therefore tesiting it's performance in real life devices becomes impossible. The proxied version is the one tested in this proyect. It requires devices with CLDC and MIDP 2.0. To accomplish this a Proxy and another protocol was developed to achieve having a small footprint in the device. The Proxy can be any JXTA peer, it must only be configured to act as a Proxy for JXME devices.

From a programming point of view, JXME is simply an API. It has only three classes: PeerNetwork, Message and Element. PeerNetwork is the main class, an instance of this class is a JXME peer. A Message is simply the class that represents messages sent to other peers and is made up of one or more Element instances. The only method that actually accessess network resources is PeerNetwork.Poll() except for PeerNetwork.Connect() which connects to the Proxy.

Because JXME has two versions, each version has it's specific architecture. This is mainly because the proxied version, as it's name implies, requires a Proxy for the devices to be able to participate in JXTA networks. The proxyless version is much more robust and takes advantage of the more ample amounts of memory and processing power. This allows the proxyless JXME devices to act inside JXTA networks directly as an ordinary peer. Fig. 1 shows this much more clearly.

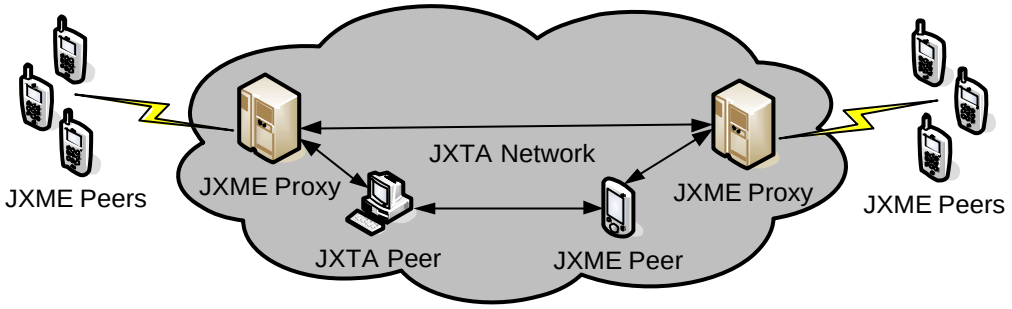

Fig. 1. JXME's architecture

2.1 Connecting to a Proxy

To craete a peer is done simply using the PeerNetwork.CreateInstance() operation. The first operation that uses network resources is PeerNetwork.Connect(). Fig. 1 shows the sequence of messages exchanged. The JXME peer first connects to the Proxy, this is the first message shown. The Proxy then responds with a message containing an element with the name "response" containing the word "pid" indicating that the message contains the peer identifier. The second element is called "peerid" which is the one that actually contains the peer id. Finally the peer asks the Proxy for a conection and the Proxy responds indicating that it will wait 36000000 milliseconds between messages before considering the peer offline.

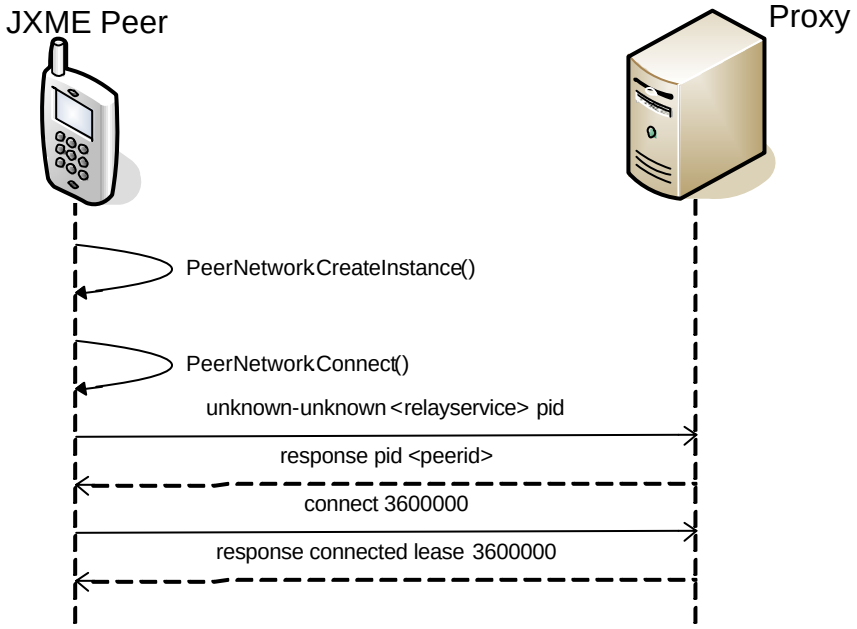

Fig. 2. Messages captured when PeerNetwork.Connect() is executed

2.2 Joining a PeerGroup

To join a PeerGroup the JXME peer uses PeerNetwork.Join(). Fig. 2 shows the messages exchanged during this operation. It can be seen that with the first message the peer is asking the Proxy to join a new group with the element “request” containing the word “join”. The element “id” contains the identifier of the group that the peer wishes to join. The “arg” element is where the password would be if the new group required authentication. Finally there is the “requestId” element which contains the identifier of this request. This means that the Proxy should include this number in it’s response for the peer to be able to identify which response corresponds with which request.

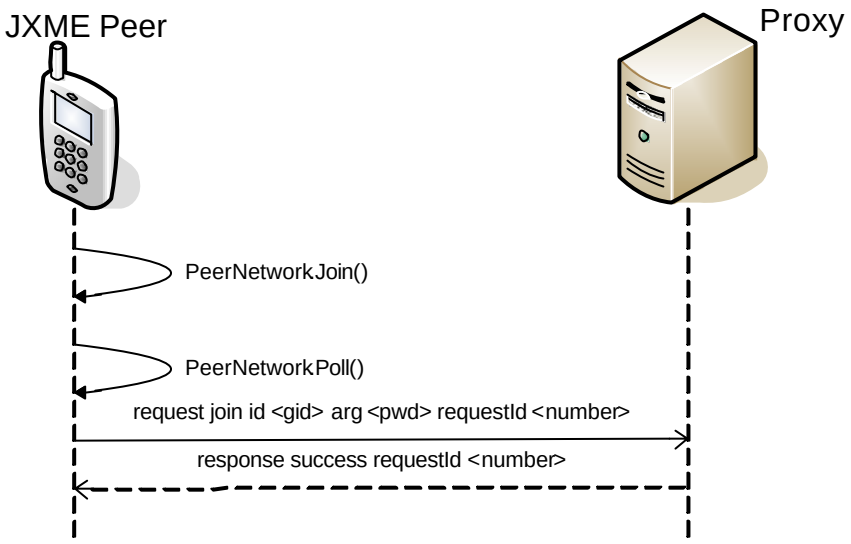

Fig. 3. The messages exchanged to join a group

2.3 Creating an Advertisement

Because of the limited capacities of the devices that are inteded to run JXME only peer and pipe advertisements can be created. However, this is enough to achieve basic communication. Fig. 3 shows the messages exchaged between the Proxy and the JXME peer to create an advertisement.

The peer first sends a “request” element with the word “create”, this asks the Proxy to create an advertisement with the given information. The “name”, “type”, any given aditional information in the “arg” element and again the request has it’s “requestId”. As a response, the Proxy sends “success” and the “requestId” and sends the information of the resulting advertisement. However, the most important part is the “id” of the resulting advertisement, this is because JXME does not have the ability to create these identifiers.

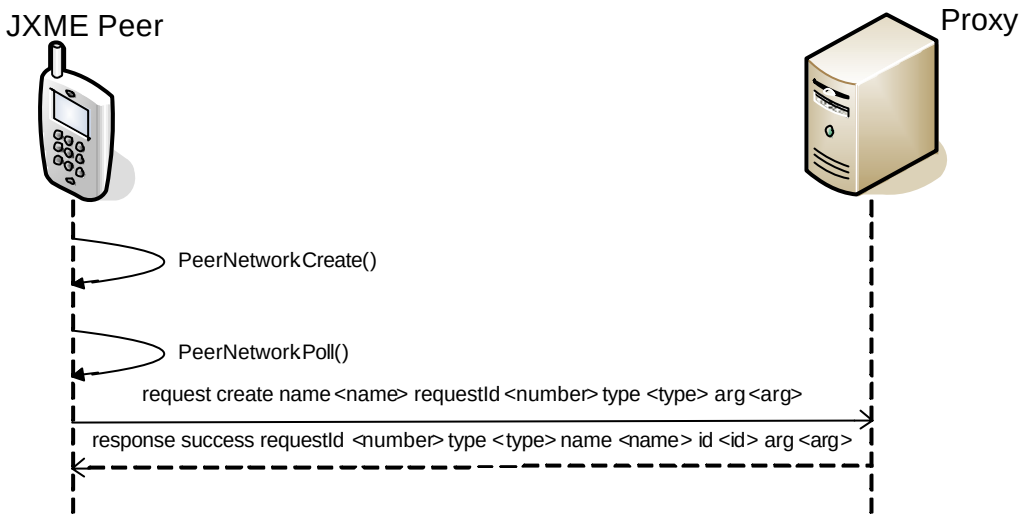

Fig. 4. Messages exchanged to create an Advertisement

2.4 Searching for Advertisements

When searching for advertisements the peer first sends a “request” to “search” for advertisements, here the peer will search for advertisements of the given “type”. The peer is interested in resources that have some text in the specified attribute given in the “attr” element. This can be seen in Fig. 4. Finally, the peer indicates the “requestId” of this operation. As a result the Proxy may send one or mosre messages with a “response” and the word “result” followed by the “requestId” of the operation, however, the maximum number of results returned is specified by the “threshold” element. The messages indicate the “type” of the advertisement found, the “name” of the resource, followed by the “id” of the advertisement and finally indicates any additional information in “arg” element.

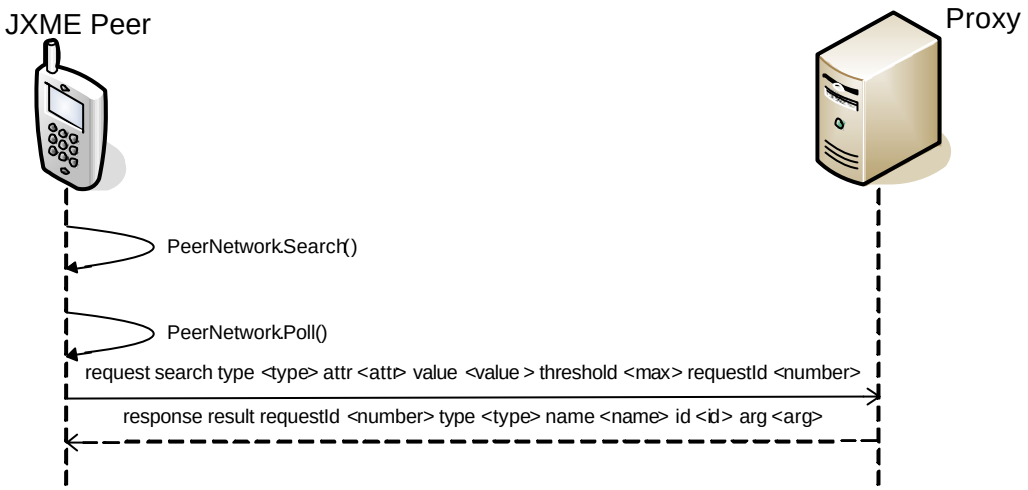

Fig. 5. Messages exchanged when searching for advertisements

2.5 Opening and Closing a Pipe for Input

To open a pipe for messages the peer sends a “request” with the word “listen”. If the operation would have been to close the pipe, it would simply contain the word “close”. Then the “requestId” element and finally the “id” of the pipe to be opened or closed. As a response the Proxy simply responds with the word “success” in the “response” element. Fig. 5 shows the messages exchanged during this operation.

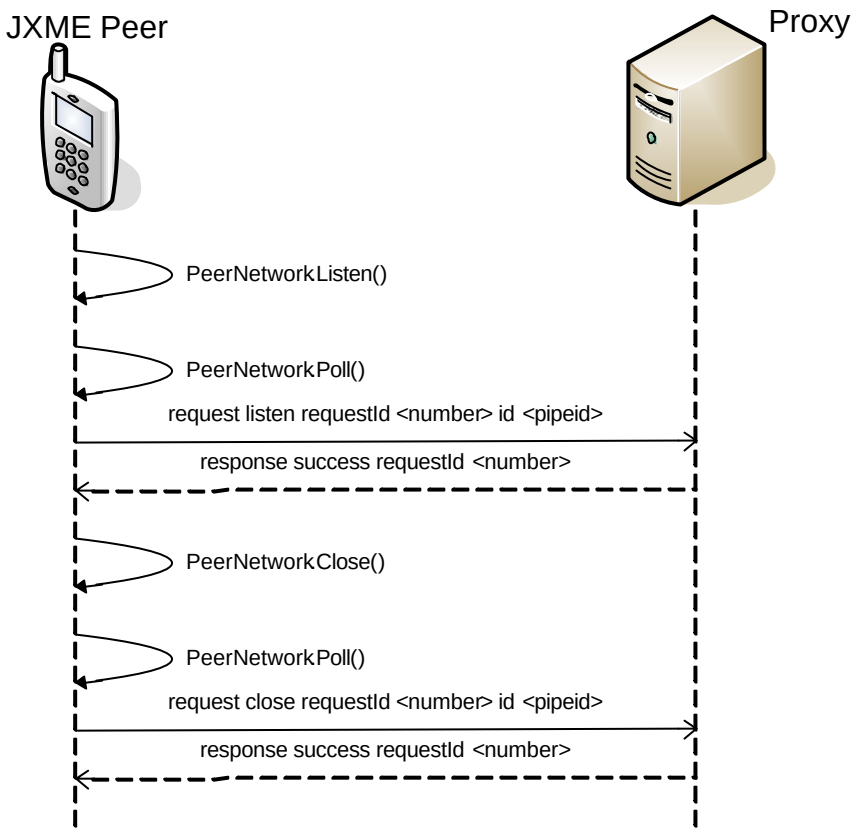

Fig. 6. Messages exchanged to open or close a pipe

2.6 Sending and Receiving Messages

The most complex operation is sending and receiving messages. Sending a message is done using PeerNetwork.Send(). Fig. 6 shows the whole process in a hipothetical (and ideal) case between two JXME peers and the Proxy.

One of the best practices in JXME programming is to periodically poll the Proxy for messages. For this reason the receiving peer is constantly executing PeerNetwork.Poll(). Therefore, in any moment the sending peer executed PeerNetwork.Send(), this operation does not use network resources immediately, it simply leaves the message ready to be sent the next time that PeerNetwork.Poll() is executed. When the message is sent, it waits inside the Proxy until the receiving peer executes PeerNetwork.Poll(). In the response of this operation (message number 6) is where the message travels to the receiving peer. Then the sending peer executes PeerNetwork.Poll() again (message number 7), the Proxy responds (message number 8) indicating the success of the operation. Messages 9 and 10 are there to show the periodicity of the PeerNetwork.Poll() operations.

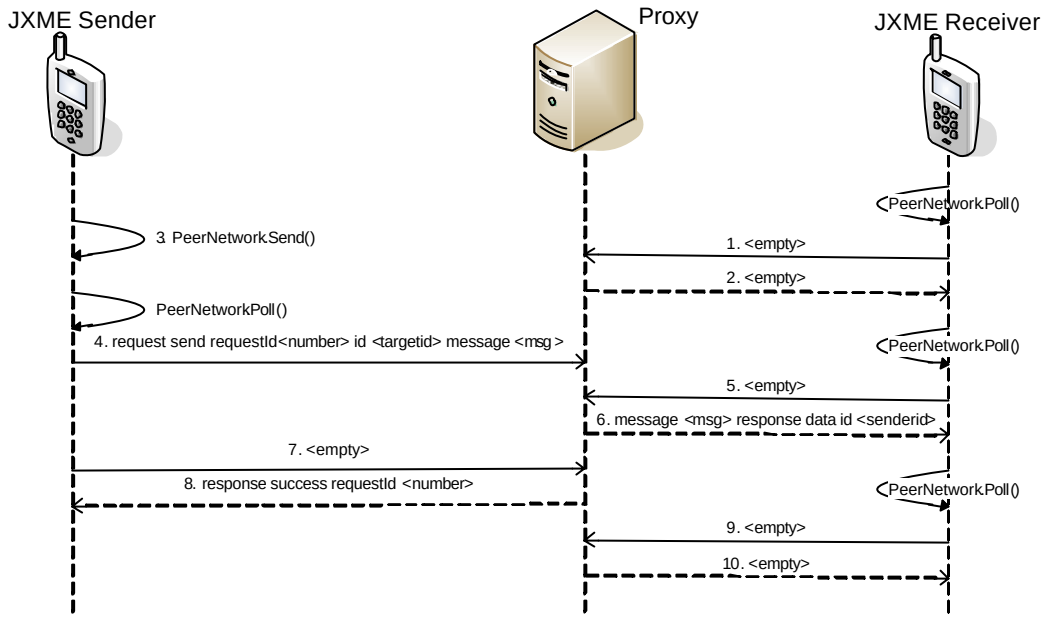

Fig. 7. Messages exchanged between the Proxy and two JXME peers to achieve sending a message from one peer to another

3 The Testing Model

Based on Halepovic and Deters' work, the idea is to create a model that would measure the performance of JXME in different environments, made up of devices and network topologies and compare these results with those of JXTA. However, both are not very comparable because they are executed under different conditions regarding comping power, bandwidth and the system's architecture. Therefore, the idea was to be able to view the differences in devices and configurations. To be able to notice the overhead of using these technologies, a simple Java servlet was developed and used to account for the latency, throughput of the system and computational capacities of the device used. The results observed with the Java servlet were called the "baseline" test described in Section 2.4. This would allow a comparison between JXME and JXTA to be independent of device and network topology and attempt to observe the differences between their expected behaviours.

3.1 Typical JXME Peer Operations' Latency

The first test is the collection of what Halepovic and Deters call "latency of typical peer operations". It is simply a measure of how much time it takes for a JXME peer to perform all the operations that a typical application would require to successfully participate in a JXTA network. However, because JXME has less functionality and a different behavior than a JXTA peer, the typical peer operations differ. Therefore it was necessary to identify the typical operations of a JXME peer. These were found to be: Connecting to the Proxy peer, indicated as "Start"; searching for, joining and connecting to the default group, indicated as "Join"; publishing the peer's own advertisement, indicated as "Publish"; opening and closing a pipe for input, indicated as "Input

Pipe"; searching for a pipe, indicated as "Output Pipe" and learning from other peers, indicated as "Learn".
This latency test could be considered to belong to the standalone tests in the "bench" project [5] because the JXME peer by itself is not considered a full peer, while the sum of the JXME proxy and the JXME device is considered a full peer [6].

3.2 Pipe Message Round-Trip Time (RTT)

Another important metric to evaluate is to see how long it takes for a minimum size message to go from one peer to another. This is similar to a ping. For this test, only application-specific messages will be taken into consideration, query-response message times are measured in the typical peer operations test described above.

The sender peer will prepare a message containing the word "PING" in its contents and send it to the receiver peer. This one would read the message and build another containing the same word and send it to the sender peer. It will read the message and finally measure the time it took to perform these operations.

3.3 Pipe Message and Data Throughput

This test sends messages one after the other as fast as possible during a given amount of time and measure the amount of application data that was successfully received. To do this, the sender peer would build and send messages to the receiver peer. The receiver peer would simply read the message and drop it. The messages would be numbered to guarantee that none are lost during the test. If this happens the test failed and must be executed again.

3.4 Baseline

These tests are executed in conjunction with a baseline test. The baseline is designed to be as similar as possible to what the tests are sending and receiving through the network. JXTA uses a Jetty HTTP Server [7] to be able to communicate through HTTP. In JXTA 2.3.4, the version used in these tests, it uses a Jetty 4.2.19. Therefore, a servlet was developed and published on the same Jetty HTTP Server. The servlet would simply send and receive messages with randomly generated text of the same length used by JXTA and JXME in each test. This way the amount of information, network speed and latency can be accounted for. This would allow these tests to measure the time that JXTA and JXME spends processing each operation.

4 Execution

The most important aspects of these tests are the network and device conditions. Considering this would show the real performance of JXTA and JXME interacting in real devices with different computational and network capabilities. The tests were run using several combinations between a cellular phone, a PDA and a PC.

The typical peer operations test only involves two entities: the JXME peer and the JXME Proxy server. The Proxy server is necessarily a PC. Therefore the test was executed from a PDA, cellular phone and emulated cellular phone. Using networks such as GPRS, Bluetooth, Wi-Fi (802.11b) and Ethernet networks.

The pipe message round-trip time and the pipe message and data throughput tests require three entities when working with JXME peers: the JXME Proxy server, a JXME sender peer and a receiver peer that may be a JXME or JXTA peer.

The devices used were: A Nokia 6230 GSM cellular phone; HP iPAQ h5500 PDAs; a PC with an AMD Athlon 2000+ processor, 512MB of RAM and a cable modem connection at 128Kbps, a Linksys Bluetooth USB dongle and a NetGear 802.11b ME-102 access point.

5 Results and Conclusions

After having executed each test, the results were gathered, observed and analyzed.

5.1 Typical JXME Peer Operations' Latency

The results of these tests are shown in Fig. 1 to 6. The charts show the averages found after executing every operation with its respective baseline test.

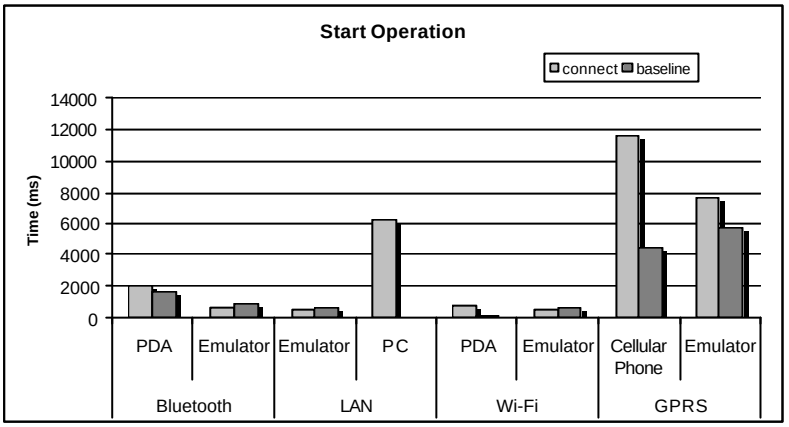

Fig. 8. "Start" operation test results

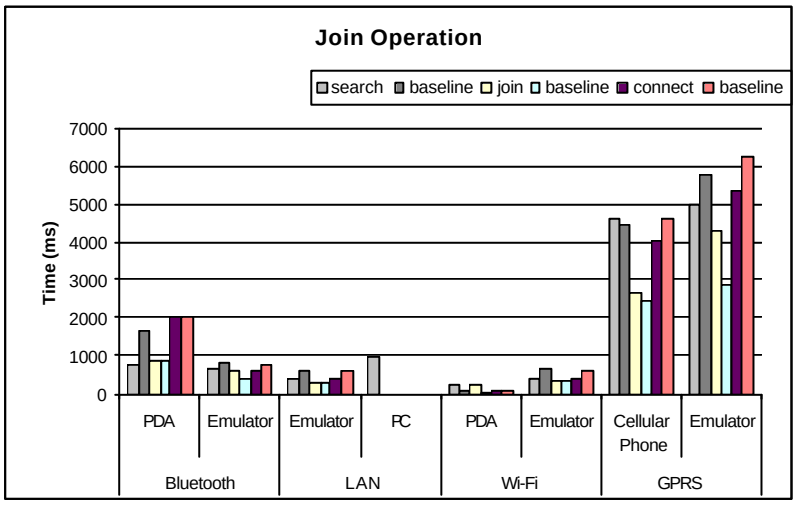

Fig. 9. "Join" operation test results

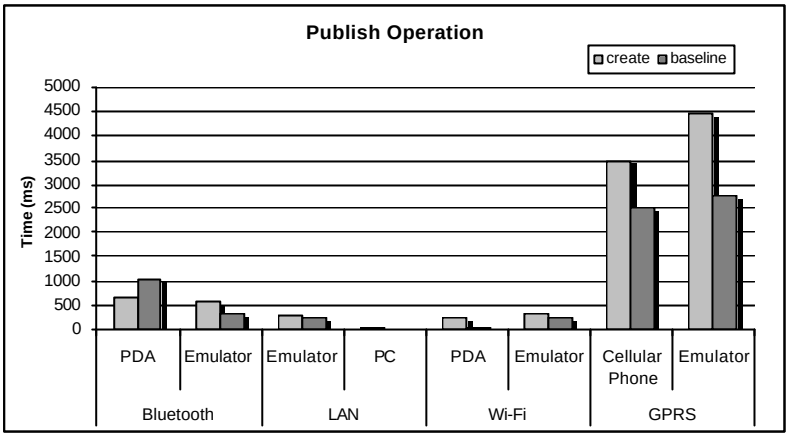

Fig. 10. "Publish" operation test results

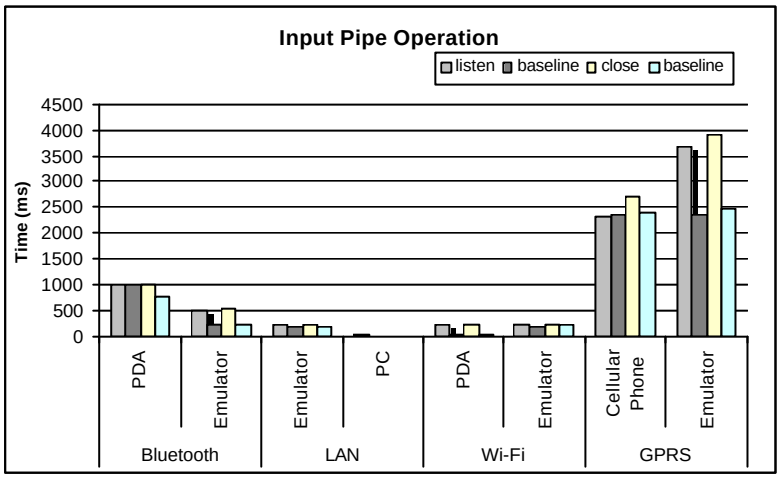

Fig. 11. "Input Pipe" operation test results

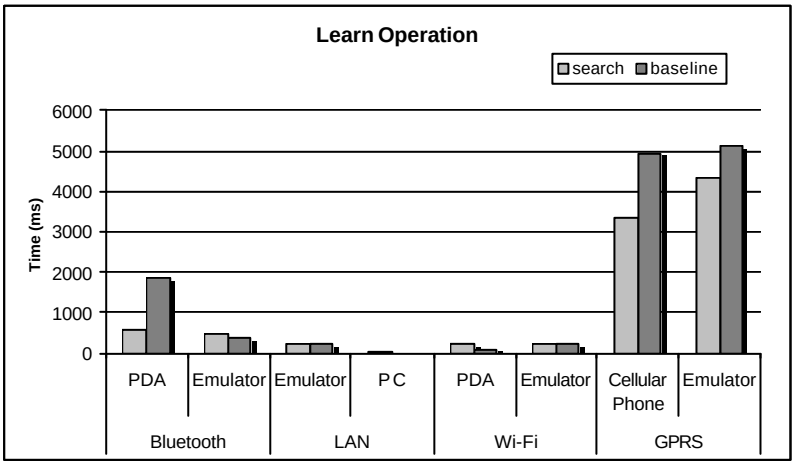

Fig. 12. "Learn" operation test results

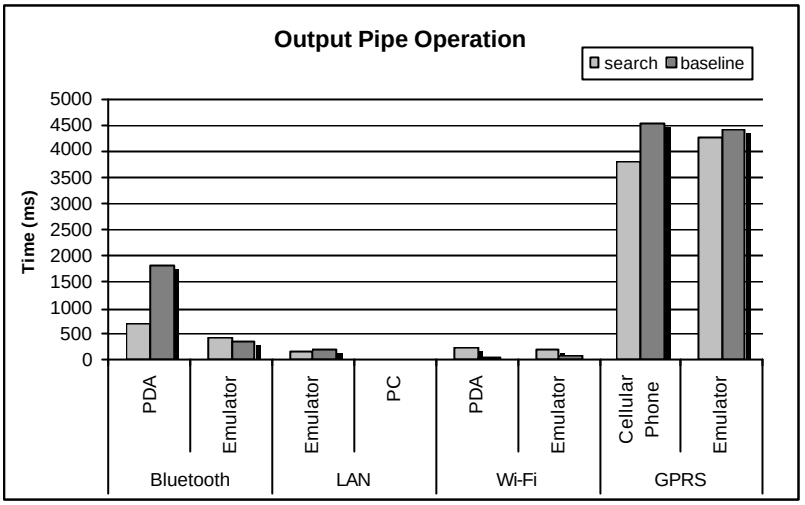

Fig. 13. "Output Pipe" operation test results

It is important to remember that the typical peer operations of a JXTA peer are very different from the operations of a JXME peer. That is why some tests do not have results. The results shown for a PC through a LAN are simply copied and adapted from Halepovic and Deter's work. This means that the results of a PC through a LAN are of an application written in Java running JXTA, while the rest are of an application written in J2ME running JXME.

The first unexpected result is the latency of connecting to the JXTA network during the "Start" test on Fig. 1. This is because this result came from Halepovic and Deters' work. They measure the time it takes for a JXTA peer to start up: loading all the classes, starting the Jetty HTTP server, connecting to a Rendezvous peer, etc. JXME, on the other hand, only has to send an HTTP message to the Proxy which it already has its IP address. Therefore the JXME peer's startup is much faster, but much more limited especially because if the configured Proxy fails, the application fails, as if it were in a client-server environment. Even if there were backup Proxies, the end-user would have to manaully enter the IP address of the backup servers and restart the entire application, because the state of the peer is maintained in the Proxy and these are not distributed throughout the JXTA network.

Another important observation is the amount of computational power required for JXME to work. This can be seen comparing the baseline tests to the JXME tests when run on a cellular phone through GPRS. Although GPRS by itself has a very high latency, the cellular phone takes quite a large amount of time processing each message and executing each operation.

However, there are operations that are quite fast and efficient. Searching for advertisements is the fastest operation JXME has, shown in Fig. 2 and 5. It is clear that the caching of advertisements is done efficiently by the Proxy, to a level that it is done even faster than sending and receiving randomly generated text through HTTP, measured in the respective baselines.

Opening a pipe is also a very fast operation, however closing them is slower. Theoretically opening a pipe would be slower than closing it. This is because to open a pipe, the advertisement must be found, read and the pipe opened, while closing it is simply destroying an object and closing a network connection. However, this is not the case in JXME.

5.2 Pipe Message Round-Trip Time (RTT)

Fig. 7 to 10 show the results of the round-trip time tests in the different environments using combinations of the different devices.

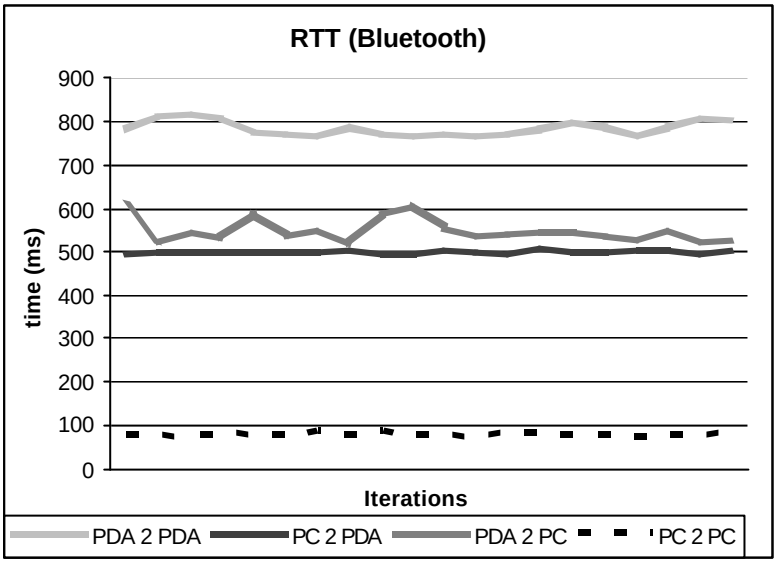

Fig. 14. Round-trip time results through a Bluetooth connection

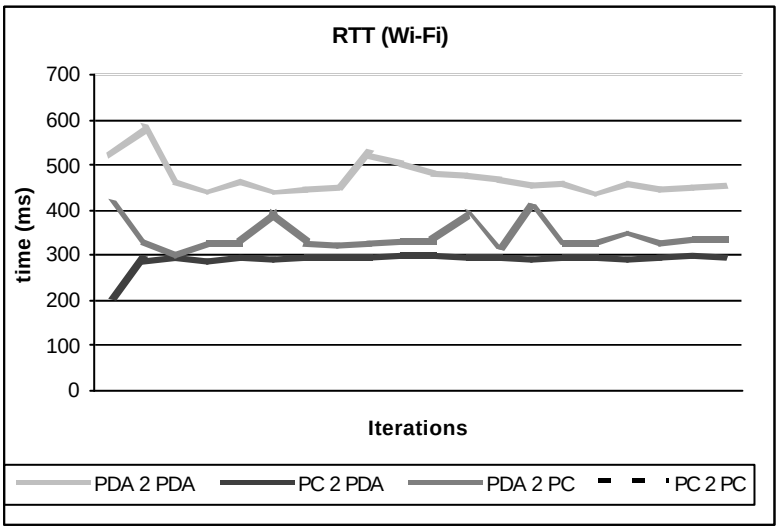

Fig. 15. Round-trip time results through a Wi-Fi connection

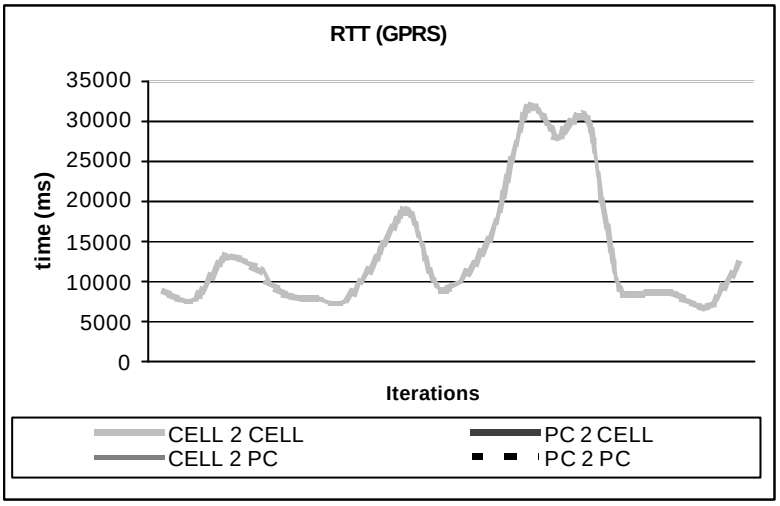

Fig. 16. Round-trip time results through a GPRS connection

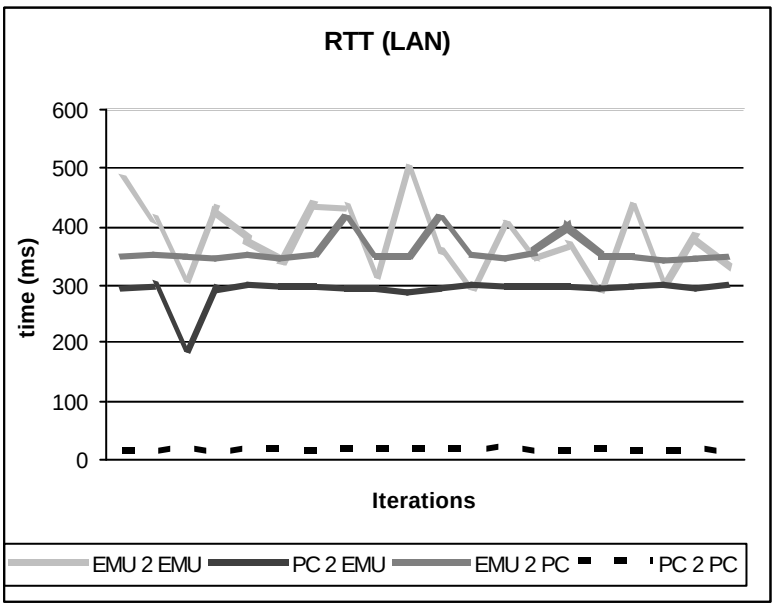

Fig. 17. Round-trip time results through a Ethernet LAN connection

The performance of JXME behaves poorly because of the way it works. While a JXTA peer simply sends the desired message and receives it asynchronously, a JXME peer has to poll the Proxy checking if there are any messages for it, very similar to the way email works. The polling is done periodically at time intervals specified by the developer. This causes the message to have to wait an unknown period of time in the Proxy until the JXME peer asks for the message to be sent to it. This behavior explains why the slowest RTT was found between two JXME peers and why sending from a JXTA peer to a JXME peer is slower than from a JXME to a JXTA peer. However, this behavior is designed this way because in GPRS networks internet access is done through a gateway that bridges the two networks, it is very similar to a NAT, where it is only possible to reply to messages and there is no way to directly contact the device inside the GPRS network from the internet.

It is also noticeable that reading a JXTA message is not a simple task therefore is time consuming. It is clear when comparing the baseline tests with the JXME tests. JXME messages are more complex than expected. They contain information that could be thought not necessary, like endpoint destination and source addresses when receiving messages from the Proxy.

Unfortunately, the GPRS network that was available to perform these tests was too unstable and had a high latency. In JXME these high latencies are managed well, but in JXTA these apperently caused the communication to fail constantly. Connectivity between the peers was available, but apparently the latency was the responsible for dropped messages. Fig. 11 shows the results of a ping command from a PC. It clearly shows the instability and high latency of the network.

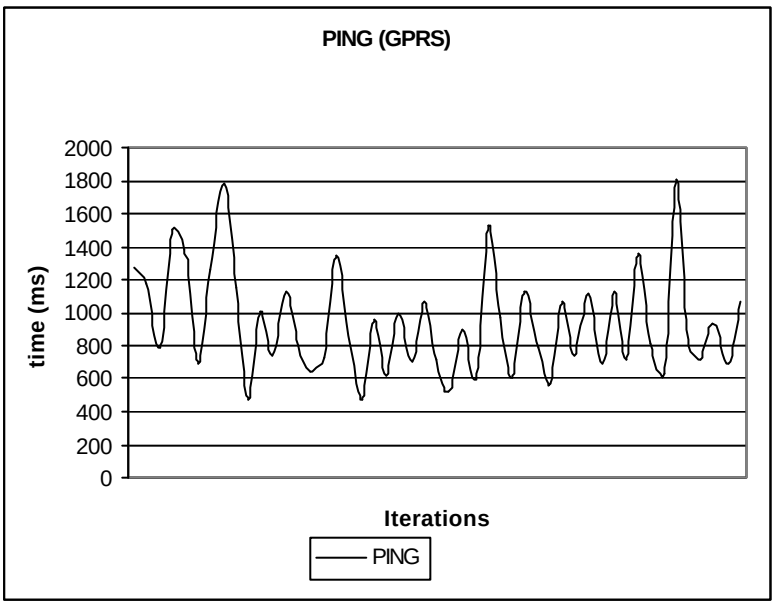

Fig. 18. Results of a ping through a GPRS connection

5.3 Pipe Message and Data Throughput

These tests in JXME cause a large amount of problems [11]. When working with messages containing under one thousand bytes of application-specific data no problems where experienced. The problems began when testing with larger messages. When a message is sent, the Proxy responds with an acknowledgement when the operation is completed. With messages over one thousand bytes of information the Proxy behaved unexpectedly, it sent the message successfully, but did not respond with the success message to the sender, or the message was simply dropped. The theoretical message size limit for this version of JXME is 64KB. For this to be able to work properly a lot of work must be done. This work is considered to be beyond the scope of this project, because the objective here is to test the performance of JXME, not the performance of specific algorithms that are required to implement functionalities that are expected to be done automatically by JXME.

5.4 Test Conclusions

During the tests some JXME implementation bugs were found. When searching for advertisements, there is a parameter that states how many advertisements are to be returned as a

result. Unfortunately, this parameter is not working properly. The Proxy returned more advertisements than the maximum that was stated to be allowed. Another bug was found when creating a new peer group from the JXME device. The Proxy did not have the appropriate module implementation advertisements available for the operation to be done. When the Proxy entered a peer group, except the NetPeerGroup, it was found that it no longer answered to petitions from the JXME device. These bugs have all been communicated to the JXTA community and are expected to be in process of being corrected.

Analyzing this experience with the CLDC version of JXME and the results of the tests, it is clear that the main reason for the design and implementation of JXME is the way how 2.5G cellular networks work. First, it is not possible to achieve direct communication between cellular phones, contrary to pure P2P systems. Second, GPRS does not have native IP support all the way to the device, allowing the device only to receive responses for petitions already made, making it imposible for devices in the internet to initiate the communication. These devices can browse the Internet because they do it through a gateway that bridges these two networks. Therefore accessing the device directly from the Internet is absolutely impossible. The developers of JXME realized this and overcame this obstacle creating the proxy server that can successfully act in JXTA networks for them.

However, JXME has a CDC version already developed and being tested actively by the community. This version would allow constrained devices to be fully functional peers in a JXTA network. For cellular phones to be able to enjoy full peer functionality the technology must wait for 3G cellular networks, where it is expected that IP support will be end to end. This version of JXME was not utilized because the main idea was to test the performance of JXME in real life conditions. JXME for CDC is still experimental because it requires CDC version 1.1, which is not yet complete.

Unfortunately, the way that JXME for CLDC is implemented not completely peer-to-peer. There is a full dependency on the Proxy. If the Proxy fails the system does not have the ability to automatically choose another Proxy for the application to continue working. The application will crash.

6 Future Work

Unfortunately benchmarking P2P systems is still an area of little knowledge of how it should be done [1]. These networks are complex, because of the unpredictability of the networks and wide variety of messages in the network, but more importantly the different capabilities of the networks and the computational capacities of the devices participating in the network.

These tests were built using the “proxied” version of JXME, which is developed for the CLDC configuration and MIDP profile of J2ME. Unfortunately, this does not take advantage of the capabilities of more powerful devices such as PDAs and some smart phones. For this reason, the JXME Project also created a “proxyless” version of the platform to be executed with the CDC configuration version 1.1 and Personal Profile version 1.0. This platform should also be tested.

These tests have all been developed and executed using JXTA Unicast Pipes [8]. These tests could be modified to test other types of pipes and see if there is a difference in their performance and reliability.

Many more questions are left unanswered with this suite of benchmarking applications. Important benchmarks like JXME Proxy stress tests are important to be verified for service providers to be able to estimate the specifications of the devices and how many machines and with what approximate specifications are needed to support a large amount of clients connected at the same time.

The way JXME is designed to send and receive messages is expensive from the network and the computation point of view. Polling a proxy for messages is a slow way of receiving messages.

Most importantly, polling the server during the whole time that the peer is online is expensive even from the economic point of view, given that most service providers charge per information transferred. This model could be improved using services such as SMS (Short Message Service) in order to eliminate the empty polling to the proxy server. This way, the proxy server could send a SMS in order to achieve asychronous behaviour on the peer.

References

[1] E. Halepovic and R. Deters, "JXTA Performance Model", Future Generation Computing Systems, Special Issue on Peer-to-Peer Computing and Interaction with Grids, Elsevier, Vol 21/3, pg. 377-390, 2004

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

[3] Project JXME Home Page, http://jxme.jxta.org

[4] JXTA Bench Project Home Page, http://bench.jxta.org

[5] JXTA Bench Project, "JXTA P2P Platform Benchmark Plan", http://bench.jxta.org/benchplan.html

[6] A. Arora, C. Haywood, K. S. Pabla, "JXTA for J2ME – Extending the Reach of Wireless with JXTA Technology", Sun Microsystems, March, 2002.

[7] Jetty HTTP Server HomePage, http://jetty.mortbay.org/jetty/index.html

[8] Project JXTA, "JXTA v2.3.x: Programmer's Guide", http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf, Sun Microsystems, 2005.

[9] Project JXTA, "JXTA Tutorials", http://www.jxta.org/project/www/Tutorials.html, JXTA Community Web Site (http://www.jxta.org)

[10] J. D. Gradecki, "Mastering JXTA – Building Java Peer-to-Peer Applications", Wiley Publishing Inc., United States of America, 2002.

[11] T. Morkved, "Peer-to-Peer Programming with Wireless Devices", University of New South Whales and Agder University College, 2005.