Management of *Peer-to-Peer* Networks Applied to *Instant Messaging*

Guillaume Doyen, Emmanuel Nataf, and Olivier Festor

Equipe MADYNES
LORIA
Campus Scientifique
54500 Vandoeuvre les Nancy
{doyen, nataf, festor}@loria.fr

Abstract. *Peer-to-peer* networking has allowed the development of new applications that couln't have been built on the client/server model. Having a good operation of such applications implies a considerable support of underlying networks and services. A typical application is instant messaging and we present an instrumentation of one of the existing one. In order to do that, we use the abstract model for such a service defined in the IETF context. Finally, this work is materialized by the instrumentation of a basic service: the presence service. We have used a particular Instant messaging application that is JXTA based : JIM. This last is built on the JXTA development environnement and allows our instrumentation to be reusable for other instant messaging applications.

1 Introduction

1.1 *Peer-to-Peer* Networking

Peer-to-Peer (P2P) systems of services and infrastructures are of a large variety, both in the nature of applications and in used models. A taxonomy of P2P networks and typical applications given in [6] details the main actual works in this domain and proposes a first step towards the generalization of some concepts. As of today, the main applications are dedicated to file sharing (like Freenet, Napster or Gnutella), grid computing (like SETI@Home), collaborative work (like Groove or MAGI) and messages exchanges between buddies (like ICQ, AIM, Jabber). Moreover, some generic platforms enable the development of applications independently of their nature (like JXTA or .NET).

The main principle of P2P networking consists in runnning an application in a cooperative and identical way among several users. The underlying P2P infrastructure aims at ensuring them that the the communication is well available. The communication models detailled in [7] can be, for some of them, very close to the client/server one, as in SETI@Home where each client executes a part of a global calculation individually and returns its result to a fixed server. Other P2P models are pure, in the sence that they do not present any dedicated server, but only entities that can play the role of both client and server (called

A. Marshall and N. Agoulmine: MMNS 2003, LNCS 2839, pp. 449–461, 2003.
© IFIP International Federation for Information Processing 2003

servent or *peer*), like in Freenet. Finally, hybrid models use dedicated servers whose role consist in grouping *peers* addresses so that a new *peer* wanting to join the community can do it efficiently. In the following of this paper, we will distinguish P2P applications from P2P networks, the first ones are using a P2P service supported by the second ones.

1.2 Network Management and P2P

The rapid success of services delivered by P2P networks implies the appearance of some potential risks for their own operation. Let's have an example with the file sharing where a major part of users are consumer first of all. In this situation, the over consumption makes resources harder to access and leads to service disruption. Moreover, danger is present too for underlying supporting networks and enterprises that foster the users (like network services in enterprises and Internet service providers). Indeed, the traffic due to P2P services can be and often already is very consequent (in particular with the exchange of multimedia documents like DivX movies) and bottlenecks of well-known servers are not so easy to locate because any user can become a server. The access facilities to these services often enable the users to bypass any authorization of the enterprise and thus to generate personal traffic accounted to the enterprise. As a consequence, it can become a potential weakness in security policies because both the tools and the exchanged content are out of any control process. Such dangers have been rapidly detected and there are several solutions to treat some of these aspects individually (for example by tracking any traffic towards a well-known *peer* server).

Functional areas of network management and P2P. Functional areas of network management can address different important points. In the following, we reuse the standard classification and apply it to P2P networks. Thus:

– **Fault managemement** that deals with discovering abnormal behavior of the network, identifying its cause and taking the appropriate actions to solve the fault, has to reconsider most of the definitions of a fault known in fixed networks or standard architectures to fit the P2P behavior.
– **Configuration management** that enable the coordination of the nodes paramaters of a network must not be located in the equipments but in the *peers*.
– **Accounting management** is in charge of associating a value to the use of resources and can be applied to the P2P networking. It consists in collecting data bout a service usage, defining its price, and finally bill the user.
– **Performance management** whose goal is to ensure a given traffic level and secondly to be able to plan the evolution of networks, devices and services, together with their configuration is dependant on the nature of the considered P2P application.

– **Security management** is in charge of the set of encoding and authentication mechanisms that allow access to resources and of those that ensure an optimal use of resources. In the case of P2P services, alternatives approaches based on trust relationship emerge.

Management solutions. There is no full managment solution for *peer–to-peer* services. Nevertheless, some architectures deals with a particular functional area of the network management. About performance management, [3] proposes to use an active network architecture to ensure performance management of a Gnutella like P2P application. [1] carry out accounting management by defining a fictitious money services that regulates exchanges between *peers*; gains are collected by the *peers* which the most participate to the service delivery.

P2P services have a strong need of management if they want to be more generally deployed and embrace new application domains. In the case of a free participation of any user, everyone benefits from delivering a way to get the whole managed correctly. In the case of private services (charged or restricted to a enterprise), the need to define policies implies the need to manage the delivered service by enforcing the specified policies.

In this context, we are seeking to propose a generic architecture dedicated to the management of P2P services. Thus, our approach consists in using a generic platform to develop P2P services. Its functional role is between the realization of services (Instant messaging, file sharing, . . .) and a common execution environnment for the all applications. A first experiment on JXTA [8] has shown its openess for management mechanisms; by using some basic functionalities available in each *peer*, these latters features an interface allowing their management.

The work presented in the remainder of this paper comes within the scope of a study to propose management models for various types of applications in the P2P world. We have chosen the Instant messaging one firstly for its simplicity enabling a very focused instrumentation, and a dependent integration of a service that is complex to isolate. Lastly, existence of several IM applications based on JXTA allows us to test the genericity of our approach and to establish some comparisons.

2 *Instant Messaging*

2.1 Instant Messages and Presence Models

The IETF has constituted a working group whose goal is to define a model for Instant Messaging applications[1]. Results of this group are a set of proposals in the draft state and some *informational* RFC. In the general model [2], two services are defined; one for the *peer* presence and another for the messages delivery. Figure 1 describes the main elements that compose these services.

The presence service (`PRESENCE SERVICE`) is charged to maintain the consistency between the state of a user and the one seen by others users. A user can

[1] `http://www.ietf.org/html.charters/impp-charter.html`

claim himself as being ready to receive messages or signal that he is away, busy or offline (only in the first two cases messages can be sent). The model proposes to represent a *peer* according to its use of the presence service; it is seen as a `PRESENTITY` when it notifies the service of its state. The users concerned by the state of an other can be informed by the `PRESENCE SERVICE` in different ways : (1) by examining (`WATCHER`) what can be done once (`FETCHER`) or regularly (`POLLER`), or (2) by notification (`SUBSCRIBER`).

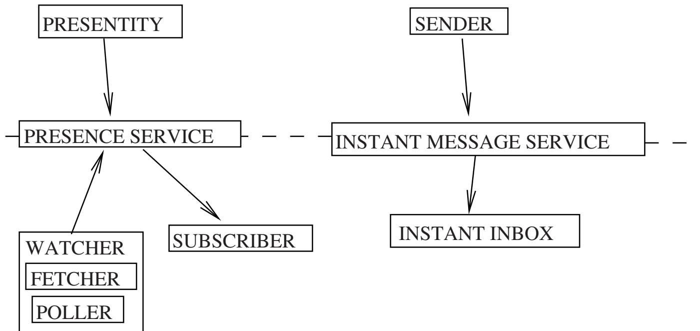

Fig. 1. IETF model for *instant messaging*

When a user wants to send a message to another one, he is described in the model as being a `SENDER` and has to use the message service (`INSTANT MESSAGE SERVICE`). This service is in charge of delivering a message into the environment of the receiver (`INSTANT INBOX`).

2.2 Measurement of the Presence Service Latency

Instant messaging application management, we aim at characterizing the quality of service delivered by all services among them, the `PRESENCE SERVICE`. This service is crucial because it ensures the consistent view that a *peer* has of the others. Whenever the service is not delivered or with too much latency, the state change of a *peer* will not be signaled to the other `SUBSCRIBER WATCHER` *peers*. Thus, a *peer* wishing to transmit a message to another one whose state appears as being able to receive messages will do it with the risk of a message loss. Conversely, when a *peer* sees another as being able to receive messages, it knows afterwards whether a message has been lost, that the recipient *peer* state has changed and that it cannot receive any message.

Different solutions are possible to measure the latency time of the presence service. Figure 2 illustrates the different elements that come in the scope of this functionality. In the case of a *peer* running as a `WATCHER` (*peer1*, according to the figure 2.a), when the *peer* 2 `PRESENTITY` signals its state to the presence service (arrow 1), it is the *peer2*'`WATCHER`'s role to launch a request towards the service

(arrow 2) and to measure the time spent to recover the response (arrow 3). This time represents the delay needed for a *peer* to obtain the state of another one.

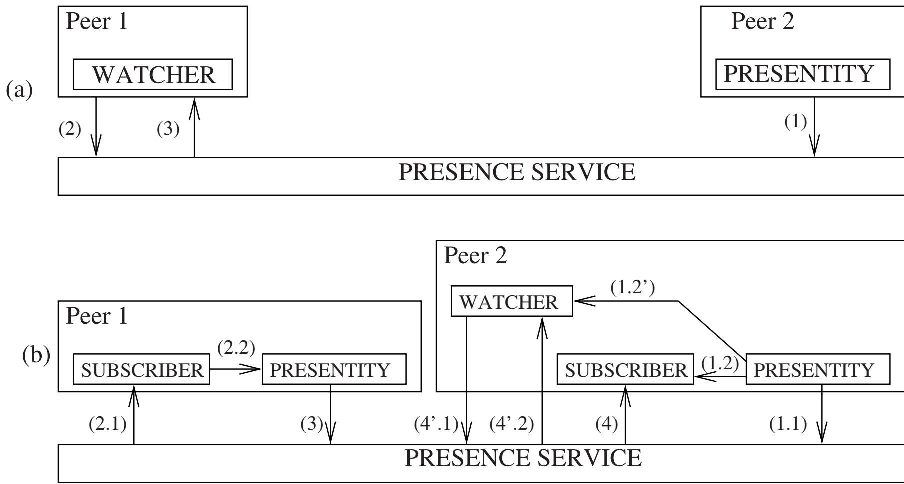

Fig. 2. Presence Service Latency Measurement

In the case of figure 2.b, one cannot measure the delay of the presence service to obtain the state of a *peer* because when the `SUBSCRIBER` of *peer*1 receives a signal from the presence service, he cannot have the knowledge of the date when *peer* 2 has notified the presence service of its state. The proposed solution for this configuration consists in measuring the time spent by the presence service to propagate the state of a *peer* towards another one, what is corresponding to a symmetric vision of the one seen just before. To realize this measurement, each `PRESENTITY` of the *peers* has to collaborate with the other constituting elements. In all cases, a *peer2* signals its state to the presence service (arrow 1.1) and the `SUBSCRIBER` of the other *peer1* is warned (arrow 2.1). In order to inform the *peer2* of the reception, the `SUBSCRIBER` has to use the `PRESENTITY` (arrow 2.2) to send an acknowledgement (arrow 3). On the *Peer 2* side, this acknowledgement can be recovered in different ways depending on whether it contains a `SUBSCRIBER` or a `WATCHER`. In the first case, reception gives immediately the measurement (namely the time spent from arrow 1.1 to number 4 one using 1.2), in the second case, it is necessary to recover the service request in the same way as in the precedent case (arrows 1.1, 4'.1, 4'.2 using 1.2'). This last measurement method is complex, implies a consequent overload of instumented elements and appears to be approximate. Indeed, transactions 1.1 and 2.1 can be done much faster than the issuance of the acknowledgement (arrows 3 and 4.x) implying a wrong increase of the RTT measurement.

2.3 Managing the Latency of Several *Peers*

Considering a global manager for the P2P application, as in the case of an enterprise or an Internet service provider, its interesting to know what are the parts of an application that can be improved or delivered with an associated service level agreement. For example, according to the figure 3, *peers p1* to *p4* form an IM application group, arrows between *peers* are as large as the time spent for the presence service to operate.

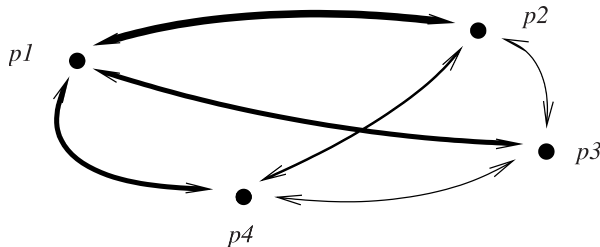

Fig. 3. *RTT* Global view of the presence service

Thus, *peer p1* seems to be more distant than the others ; *p2* and *p4* have slower links than with *p3*. Consequences of these remarks can be, at the underlying network level, to verify whether there is no element decreasing the traffic rate and to differently bill according to the presence of some of them (e.g. *p1* presence decreases the global service cost due to the decreasing of quality).

3 JIM

In order to build an instrumentation of the presence service of an IM application, we have used the JIM freeware (JXTA *Instant Messaging*)[2]. This one fits the IETF model presence by using `WATCHER/ POLLER` *peers*. Secondly, its design built over JXTA allows us to compare differents IM implementations over this environnement and to prepare an instrumentation of JXTA itself.

We chose JIM, because contrary with other IM applications it uses a P2P model. Indeed, AOL Instant Messenging or ICQ uses a client-server model while Jabber presents a functionment very similar to the mail: clients under a domain send and receive instants messages through a well-known server, and the routage between the different domains is done by a server-to-server communication.

3.1 States in JIM

A *peer* running JIM can present several states: *online* when it is ready to receive and answer messages, *busy* when it is ready to receive but cannot answer

[2] http://visis-www.informatik.uni-hamburg.de/jim

immediately, *away* when it is ready to receive but will answer in a long time (more than in *busy*). Finally, the *offline* state means that the *peer* is unable to receive messages ; any attempt at sending will result in a refusal from the message service.

3.2 JIM and JXTA

Figure 4 locates the JIM application in relation to JXTA, which provides several building blocks and APIs to build P2P applications. Among these ones, *pipes* are logical channels through which a *peer* can send data towards one or several *peers*. The communication between *peers* can be effectively done insofar the *peers* belong to the same *peerGroup* (which represents a set of *peers* in JXTA). Security and monitoring services of the JXTA core are not directly used by JIM, but are used by other JXTA elements in an implicit way.

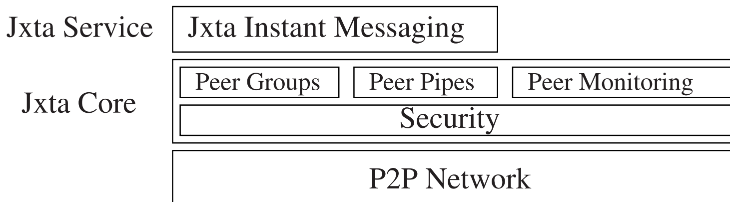

Fig. 4. JIM / JXTA Architecture

JIM uses JXTA to create a users *peerGroup* for the IM application and *pipes* to implement messages and presence services. Figure 5 illustrates the presence service mechanism; one can first notice that each *peer* contains the whole set of presence management elements. *Peer p1* owns a list of *peers*, called the *buddylist* connected to the JXTA *PeerGroup* created for JIM and uses a *pipe* towards each of them. The `PRESENCE SERVICE` is not delegated to a centralized persistent element. In this way, the operational model is the following : at regular intervals, the Poller sends a `PING` message to each *peer* belonging to the *buddylist*. Each *peer* replies to this message by issuing a `PONG` containing its state.

According to the IETF model, the *peer* sending such requests plays the role of `POLLER` towards the presence service that is represented by the reply of each *peer* contacted by the *pipe*. The model instantiation is represented in figure 5. Entities *Status Updater, Controler and User* are software parts of JIM that respectively plays the role of `WATCHER`, `PRESENCE SERVICE` and `PRESENTITY`. The *StatusControler* plays a double role because it deals with both `PONG` messages (`WATCHER`) and `PING` ones (`PRESENCE SERVICE`).

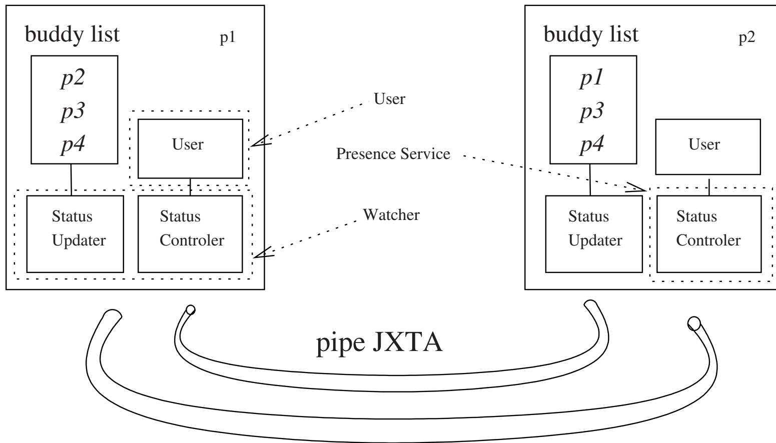

Fig. 5. JIM *peers* and JXTA *pipes*

4 Instrumentation

In order to measure the service presence performance, we chose to instrument the RTT of a `PING(Buddy)/PONG(BuddyStatus)` exchange. The collected result represents the time a *peer* has to wait to be notified of one buddy status. We use the ARM[3] (*Application Response Measurement*) java API and implementation to instrument these exchanges as transactions. Each transaction record will be registered in a management framework, namely the JMX (*Java Management eXtension*) java API.

4.1 Application Response Measurement

ARM is designed to evaluate service levels of single-systemss and distributed applications by measuring the availability and performance of transactions.

Figure 6 illustrates the ARM architecture. In order to carry out the measurement of a transaction, an application has to define transaction objects that best fit the required measurement. For example, some long-lasting transactions may have to be regularly updated while other ones may require additionnal metric informations. The entity responsible for creating and fostering transactions is the ARM Producer. The ARM agent is responsible for setting record objects properties for each finished measurement and communicating them to any ARM consumer that has subscribed to this type of measurement. Indeed, to catch the records of transactions, a consumer populates a pool with empty ARM record objects. Thus, when a transaction is finished, the ARM agent extracts an object from the pool, sets its different properties and pushes it in the queue. By calling a `get` method, the consumer can in its turn extract the record from the queue, exploit it and put it back in the pool for any further use.

[3] http://www.opengroup.org/tech/management/arm/

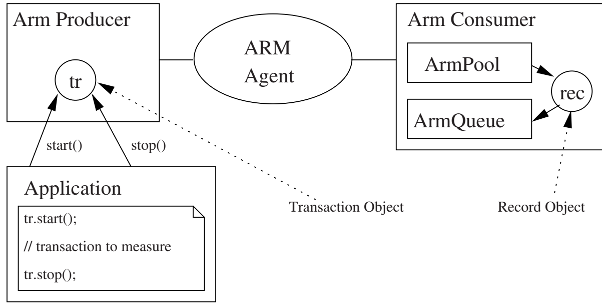

Fig. 6. The ARM architecture

4.2 Definition of a Transaction

We have defined a transaction as a `PING(Buddy)/PONG(BuddyStatus)` exchange between two buddies. Nevertheless, a simple response time value didn't provide us with enough information for our management objective. Thus we have used the metric definition feature of ARM to cope with this lack of information.

ARM allows the specialization of transactions by the use of metrics. A metric is a data that can provide an additionnal measurement (e.g. the size of a transaction) or any informational value (e.g. an indentifier). For our instrumentation, we have equipped our transactions with the following metrics :

- *the receiver identifier* : indicates the ping receiver;
- *the receiver status* : indicates the status of the ping receiver;
- *the transaction status* : is a boolean information that indicates if the measurement has stopped by receiving a PONG response or by being interrupted of a timeout signal managed by the ARM producer;
- *the transaction correlator* : uniquely identifies a PING/PONG exchange between two *peers*.

4.3 The Measurement Process

The way we measured the time spent for a PING/PONG exchange is described on figure 7. At first, in the source code of JIM we have add calls to `start` and `stop` methods respectively immediately before sending a PING message and immediately after receiving a PONG one. The `start` and `stop` methods are called on a Transaction Pool object that allow us to measure several transactions simultaneously. Moreover, the application cannot know which transaction object is actually started or stopped. When a transaction object has been started and stopped, for each subscribed consumer, the ARM agent extracts a free record object from the pool, sets the response time and the different metrics, and finally stores it in the queue. Considering the exploitation of the results, in the case of the CVS consumer, it gets all the record object properties, write it in a CSV (*Comma Separated Value*) file and put the record object back in the pool. In the

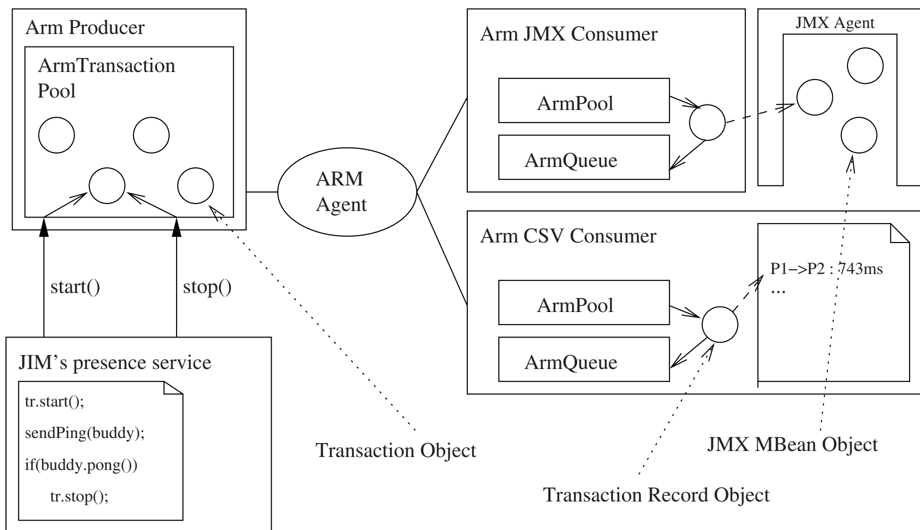

Fig. 7. The instrumentation process

other hand, in the case of the JMX consumer, for each record extracted from the queue, it creates a new standard MBean object containing all the record object properties and registers it in a JMX agent. These instances are accessible by contacting the agent (by using a HTML adaptor or a RMI connector provided with the JSR 160[4] JMX implementation, for example) or can be serialized to be transported in a JXTA service dedicated to the management (see [8]).

4.4 Correlation of the Measurement

In our instrumentation, we have to measure the time spent for the exchange of a PING / PONG message couple. Ping messages are sent every 10 seconds. PONG messages are received in an unbounded time. Thus, whenever we had used a single transaction we could not have had the certainty that a received PONG message was the one corresponding to the last PING emitted. We coped with this problem by (1) creating a transaction pool that contains enough transaction objects so that several PING can be sent simultaneously, and (2) provisioning each transaction and PING/PONG couple with a correlator that ensures that the receipt of a PONG message will stop the corresponding transaction.

4.5 Observed Results

Figure 8 show the response times of PING/PONG exchanges between two *buddies*. They have been obtained from a CSV file containing measurements on a

[4] http://www.jcp.org

100Mb Ethernet local area network. A *rendezvous peer* located outside the LAN (provided by the JXTA Developers community) was used. Our experiment consisted in measuring the response time in a permanent state, after a transient phase where *peers* discover themselves for the first time. Figure 8.a, shows the occurence of peaks that are due to the underlying P2P network. A instrumentation of JXTA will give us explanations about these peaks. Nevetheless, according to figure 8.b, we may assume that the response times present a certain regularity; the average notification time of the state of a *peer* to one of its buddy is about 850 ms with a standard deviation of 300 ms.

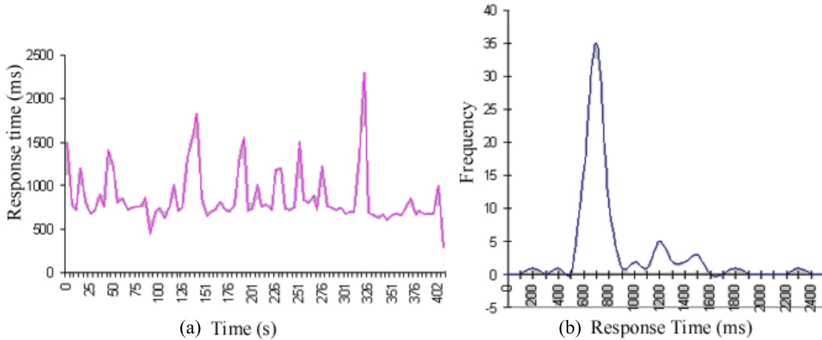

Fig. 8. Response time and distribution measurement

This first experiment was not to have a complete statistical study of the performances of a P2P application and the underlying virtual network (complete studies can be found in [5] and [4]).

4.6 Overhead Analysis

Instrumenting JIM has implied an overhead that we have always tried to minimize. Nevertheless, we have characterized it in terms of :

- *Network traffic* : Our instrumentation has required to add a correlator, represented by a unique long value, in the PING/PONG messages exchanged between two *peers*. The XML encoding specified by JIM has implied the definition of a new couple of tags, namely `<correlator>` and `</correlator>` for the insertion of our correlator in the messages. Thus, we may assume that, in terms of network traffic, the overhead introduced by our instrumentation doesn't exceed 50 bytes, which represents about 10 percent of the total size of a PING message.
- *Memory space* : the memory use of our instrumentation is due to the record of the results in a CSV file. Each result is about 150 bytes large. In this way, the file will grow at the speed of 60 ko per hour. This growth seems to be reasonnable. Moreover, further mechanisms of data compression, or filing

may be deployed in case a contiuous and very long time of use of JIM, but they are out of scope.
Now, considering the read-write memory overhead, our instrumentation increases the used memory space of the application of about 20 percent, with a total of 40 Mo. This huge overhead is due to the different threads running simultanously (ARM agent, Transaction producer, 2 transactions consumers and the JMX agent).
– *Processing time* : We have tried to have the simplest code as possible in terms of code algorithms and number of threads inducing the smallest overhead of processing time.

5 Integrating the Measurements in a Management Framework

We have chosen to use the JMX framework in order to integrate our measurement in a management architecture. JMX is a Java infrastructure that enable the management of Java applications. At the instrumentation level, JMX uses managed Beans (*MBeans*) as bases components of the architecture. A *MBean* represents a Java managed object that implements a specific interface. At the agent level, a JMX agent is container of *MBeans* and allow a remote application to access them by using a dedicated protocol adaptor or a connector.

5.1 Use of Record Objects as *MBeans*

In our experiment we have focused on the performance of the presence service of JIM and especially on the PING/PONG exchanges. Nevertheless, others aspects of JIM may require performance measurements implying the definition of multiples types of transactions and thus of *MBeans* interfaces. For this reason, we have chosen to use dynamic *MBeans* in our framework. It allows *MBean* objects to present a dynamic interface corresponding to the type of the measured transaction.

When launching JIM, we populate the pool of the JMX consumer with objects that inherit from `ArmRecord` and implement our dynamic *MBean* interface. When a record is present in the queue, we extract and register it in the JMX agent. Besides, when a record has to go back in the pool, we simplely unregister it from the JMX agent enabling it to be reused later.

5.2 The Naming Scheme

Each *MBean* object registered in a JMX agent is named according to a convention thats presents the following form :

```
Domain : attr1 = value1, attr2 = value2, ...
```

where `Domain`, `attr`*i* and `value`*i* are strings characters. In our instrumentation, we use the the following convention to uniquely indentify a *MBean* representing a transaction measurement :

- the domain identifier is the name of the *peer* currently running JIM;
- attributes and values are defined as:
  - dest = the name of the ping receiver;
  - corr = the value of the correlator used in the PING/PONG exchange.

6 Conclusions and Future Works

In this article, we have presented an instrumentation of a simple P2P application: the instant messaging. From an abstract model we have expressed how to measure the quality of a particular service used by such an application : the presence service. A well operating of this service ensures the quality of the application and needs to be managed. By registering measurement results in a JMX agent, we allow a global management application to manage the different *peers* that participate to the service. Our future works will consist in extending the definition of *peers* characteristics that need to be managed independantly of any application. We aim to define an abstract model for P2P services that enables us to design manageable *peers*. These ones will present a standard management interface that a centralized or distributed manager could use in order to manage a particular service. Finally, we plan to instantiate this model on JXTA and use JMX as a management framework.

References

1. P. Antoniadis and C. Courcoubetis. Market models for p2p content distribution. In *AP2PC'02*, July 2002.
2. M. Day, J. Rosenberg, and H. Sugano. A model for presence and instant messaging. RFC 2778 (Informational), IETF Network Working Group, February 2000.
3. H. deMeer and K. Tutschku. An application-level active networks based architecture for the performance management of peer-to-peer services. OPENSIG 2001 Workshop: Next Generation Network Programming, September 2001.
4. T. Hong. *in [7]*, chapter Performance, pages 203–241. Number 14.
5. R. Matei. Peer-to-peer architecture case study: Gnutella network. http://people.cs.uchicago.edu/m̃atei/PAPERS/gnutella-rc.pdf.
6. Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Laboratories, Palo Alto, March 2002.
7. Andy Oram, editor. *Peer-to-peer: Harnessing the power of Disruptive Technology*. O'Reilly & Associates, 2001.
8. Radu State and Olivier Festor. A management platform over a peer to peer service infrastructure. In *8 th. IEEE Intl. Conference on Telecommunications*, Tahiti French Polynesia, February 24 March 1 2003.