Towards a Peer-to-Peer Content Discovery and Delivery Architecture for Service Provisioning

Emir Hammami
LAAS-CNRS
7, Avenue du Colonel Roche 31077 Toulouse Cedex 4, France
emir.hammami@laas.fr

Abstract

Service provisioning includes systems and infrastructures designed for the sharing of services and any other data between users. Before services can be used, they have to be published, discovered and downloaded. These operations are supported by content distribution systems. Moreover the Peer-to-Peer (P2P) approach circumvents many problems of the Client-Server approach and offers an attractive alternative to exchange services over networked nodes. In this paper we first examine issues related to service provisioning systems. Then we present the design and the implementation of a platform allowing users to share contents like services in a P2P network. The platform is composed of a set of modules that provide means to advertise contents and discover others available on the network. The realized API can be integrated into more sophisticated application-level services. We conducted several experiments to assess the performance of the platform. Results show the efficiency and the effectiveness of our approach.

1. Introduction

Electronic services are software entities that perform actions on behalf of other entities [1]. Before services can be used, they have to be published, discovered and downloaded. These steps characterize the life-cycle of a service:

- **Service advertisement**. In order to inform other users about the existing services, the service provider creates a service description that supplies a list of attributes and properties about the service. Then it makes them publicly available.
- **Service discovery**. Service discovery is an important feature. It is the process of locating, or discovering, one or more services. At the end of the discovery, the requester user receives a list of available service descriptors that satisfy his need.
- **Service delivery**. Once a user knows that an interesting service is available, then it can starts a download operation to transfer a copy of the service locally.

In addition we can find other issues such as automatic service selection. This feature is especially benefit in case of download fail; it allows making intelligent choices about the location in order to retry the service delivery from other service providers. The different concepts discussed above are supported by content distribution systems [2]. There are mainly two models: the Client-Server (C/S) model and more recently the Peer-to-Peer (P2P) model.

In C/S model, a dedicated server stores contents or maintains directories of available contents. Thus, to access a content, the user asks the server which returns relevant contents matching the user query. The transfer of a content is done from the server to the user. Advantages of this approach are: high recall[1] and high precision[2] of the discovery, and efficient security policies. However, there are also negative aspects such as the large network load around the server that can causes bottleneck, and the inefficient use of resources (such as storage space and network bandwidth) available on client nodes.

In P2P model, each user shares own resources from its peer and discovers others available on the network. Resources can be physical resources, such as CPU cycle, disk space or network bandwidth, as well as, logical resources, such as services or any form of knowledge [3]. In contrast with C/S model, a P2P model is characterized by these aspects:

- Communications and exchanges between nodes are performed directly without relying on intermediate nodes.

[1] A standard measure in information retrieval. It describes the proportion of all relevant contents included in the retrieved set [4]. The highest value of recall is achieved when **all** relevant contents are discovered.

[2] A standard measure in information retrieval. It describes the proportion of a retrieved set that is relevant [4]. The highest value of precision is achieved when **only** relevant contents are retrieved.

– Nodes are autonomics and heterogeneous. They are typically personal computers or any other device with a “digital heartbeat” like PDAs.
– A node can have one or tow roles at the same time (provider/requester).
– Nodes are volatiles. Joining and leaving the peer group is frequently resulting in a dynamic network.

We argue that P2P is the ideal platform to support service provisioning requirements due to its inherent characteristics. It allows developing direct forms of information exchange between users without using central servers [5]. The expected benefits are the distribution of storage and bandwidth load, as well as robustness [6].

In this paper, we present the design and the implementation of a platform allowing users to share contents in a P2P network while discovering others. The platform provides answers to service life-cycle issues thanks to a layered architecture. The first layer encapsulates basic primitives used by any P2P application. We use JXTA [7] for this purpose. The second layer composed of generic modules hides the complexity of the underlying P2P infrastructure. It offers means to publish advertisements and to send discovery queries in order to retrieve relevant contents. Such contents can refer to services or any other data. The top layer, the application layer, consists of customized applications that can be developed more easily and efficiently.

The paper is structured as follows. We first investigate features offered by our platform. Then we describe the overall architecture with the details of each module. Afterwards, results of an experimental evaluation of content discovery and content delivery are presented. Section 5 discusses some related works. Finally, section 6 presents our conclusions and ideas for future works.

2. Features addressed by our platform

We contribute to the development of efficient service provisioning systems by proposing a generic platform for content discovery and delivery. Services can be implemented as contents and shared between users in a P2P fashion. This platform supports the following features:

2.1. Local content management

A content can be anything to share, like a multimedia document or a simple file. Each peer node owns a set of contents stored on the local file system. A content is represented by a unique Id and a content descriptor, which provides meta-data about the content, such as its name and other attributes. This information is stored on a repository.

As we focus on generic platform, our system allows a developer to create customized class of contents. Each class of content belongs to a particular content model. Access operations to the repository are the same for different class of contents. We then define a generic module to support these operations.

2.2. P2P Network Access

This feature aims to assist the developer to achieve common P2P activities at a high level while hiding the complexity of such operations at a low-level. For example, the platform allows a node to join the existing P2P network (or to start a new one).

2.3. Sharing of contents with other peer nodes

The framework provides generic facilities for content sharing between peers. Each peer node decides what content it wishes to share. Based on the content descriptor, a content advertisement is created for each content. Unlike the centralized model where content advertisements are stored centrally on a dedicated server, in our approach, the owner peer maintains a local cache of available advertisements. When an advertisement is published, it is stored, and indexed in this local cache. This results in a pure decentralized P2P topology. When a peer is disconnected from the network, its contents will become also unavailable.

2.4. Searching of contents on the peer group

Shared contents that have been made available can be searched by other nodes. Using the platform, users can interrogate peers currently connected, and search for contents by providing keywords. The search facility follows a decentralized schema where the query is broadcasted within the peer group.

After receiving the discovery query, a peer first verifies its cache seeking for advertisements that match the query. If it has the right advertisements, it will send a discovery result back to the peer that made the query.

2.5. Retrieving content descriptors and related contents

Our platform provides a protocol based on query/response messages to carry out the transferal of content descriptors and related contents between the requesting peer and the requested peer.

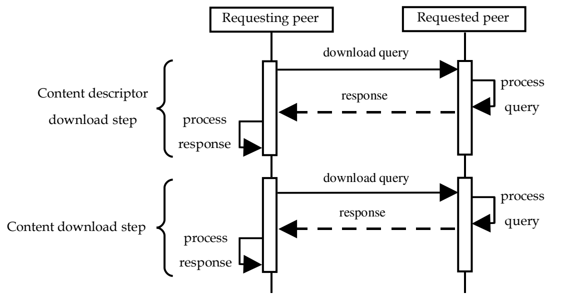

Figure 1. Download steps

As illustrated in Figure 1, the protocol consists of two steps: the first one aims to download the content descriptor in order to obtain information about the content before getting it. The second step aims to download the content itself.

At the end of the discovery phase, the requesting peer obtains a list of content advertisements. By using the address field contained in the advertisement, it sends a download query to the requested peer. This later returns the required content descriptor. By parsing the descriptor, the requesting peer determines if it refers to a relevant content. In that case, it initiates another download operation. The requested peer receives the query and returns the specified content. Messages exchanged have the same structure but data encapsulated on each message is different. More details are given in the next section.

3. Towards a common architecture

This section starts with a general overview of the system architecture followed by a more detailed presentation.

3.1. Global architecture

Our platform is a generic P2P content distribution system on which P2P applications can be developed easily and efficiently. The hole system is built up around a collection of peers that are networked together. Each peer is a single participant that operates independently and asynchronously from all other peers. It can take one or several distinct roles:

- It can adopt a *reactive provider* role if it stores contents and answers to download requests,
- It can play a *proactive requester* role if it starts download requests,
- Finally, a peer can act both as *reactive provider* and *proactive requester.*

3.2. Node architecture

Figure 2 outlines the basic structure of a node. It follows a layered architecture:

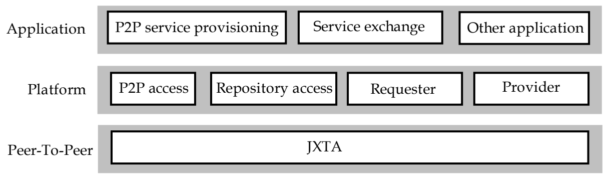

Figure 2. Node architecture

- **P2P substrate**. This layer represents the underlying P2P technology that offers networked-level functions for discovery, grouping, and communication between peers.
- **Platform layer**. Our main contribution concerns this layer. It provides an abstract layer to rapid P2P application development. Four modules compose this layer: The first module, repository access module, is a persistent data management system that facilitates storage and manipulation of contents stored at the node. It deals with the first feature identified in the previous section. The next module called P2P access module, includes high-level methods that cover features 2, 3 and 4. Requester and provider modules handle all communication aspects within the P2P network to perform the download operation according to feature 5 of section 2. Thanks to these modules a developer do not need to understand how the P2P substrate operates.
- **Application layer**. In this layer, developers exploit the proposed modules as building blocks for customized P2P applications. Service provisioning systems can be developed at this level.

3.3. Peer-to-Peer layer

This is the bottom level of our architecture. It encapsulates low-level services required by any P2P application. We use JXTA for this level. JXTA is a platform independent P2P framework, created by Sun Microsystems in 2001. It was designed to allow a wide range of devices (PCs, mainframes, cell phones, PDAs) to communicate together and to exchange messages in spite of the network topology.

The main goal of JXTA is to provide a set of core protocols to accomplish the most common P2P tasks such as resource discovery, membership service, and communication between peers that involves data exchange and message routing. A peer in JXTA can join and leave a peer group, which is a collection of peers having common interests. In order to exchange messages and data between peers, JXTA uses pipes as virtual communication channels.

JXTA also introduces the concept of advertisement in order to describe any entity to publish in the peer group. Concepts offered by this project are useful for our work; in particular advertisements are used to announce contents.

3.4. Platform layer

Platform can be viewed as a layer on top of JXTA that provides additional abstractions. Through a set of generic classes and interfaces, this layer supplies modules that simplify the creation of Peer-to-Peer content sharing applications. In the sequel, we detail this layer. The presented API is slightly simplified for clarity.

3.4.1. Repository access module. This module exports the following APIs:

– *Content* represents a content stored on the local file system. It is composed of two attributes: *ContentID* and *ContentDescriptor*.
– *ContentID* contains a unique 128-bit MD5 chechsum generated from the content descriptor.
– *ContentDescriptor* represents the descriptor of the content. It is composed of a tree of *Element* items.
– *Element* can be either a simple element composed of a key-value pair accessed by getKey() and getValue() methods, or a composite element formed by other elements (both simple and composites elements). G*etChildren(String)* method allows to access to the first element having the key field equals to the value transmitted in parameter.
– *ContentManager* offers the following operations to manage contents: *getRepository(), createContent()*, *saveContent()*, *removeContent()*, and *getContent()*.

3.4.2. P2P access module. The main goal of this module is to offer methods according to these topics:

– **Group Management** allows a node to be inserted to or to be retired from the Peer-to-Peer network. *P2PNetwork* abstracts a peer group while *P2PNetworkAdvertisement* abstracts the advertisement associated to this peer group.

Table 1. Group management methods

Methods	**Description**
initPlatform(String peername, String user, String password)	Initialise the P2P application
createGroup(P2PNetwork parent, P2PNetworkAdvertisement advert)	Create a peer group
joinGroup(P2PNetwork group, String user, String password)	Join a peer group
leaveGroup(P2PNetwork group)	Leave a peer group

– **Advertisement creation.** A group, a channel and a content are identified by an advertisement. *GroupAdvertisement*, *ChannelAdvertisement* and *ContentAdvertisement* classes are instantiated with appropriate parameters in order to create advertisements. Since our proposition supports various content types and to avoid create one *ContentAdvertisement* class per type of content, we use only one advertisement which contains common information like content identificator, the name, the type, and keywords. Thus P2P access module offers *createContentAdvertisement()* method in charge of extracting data from the content descriptor to include in the *ContentAdvertisement*. This generic method will be customized according to each content type.

Table 2. Advertisement creation methods

Methods	**Description**
createGroupID(P2PnetworkID parentGroupId, String clearTextID, String function)	Create a peer group identificator
createGroupAdvertisement(P2PNetwork parent, String name, String description)	Create a peer group advertisement
createChannelID(P2PNetworkID groupID, String clearTextID, String function)	Create a channel identificator
createChannelAdvertisement(ChannelID channelId, String channelname, String type)	Create a channel advertisement
flushAdvertisements(P2PNetwork group, int type)	Remove local advertisements

– **Advertising.** The advertisement phase depends on the underlying P2P layer. In JXTA, there are two possibilities: (i) local publishing which means saving advertisements in a local cache that the provider parses when it receives a discovery query. (ii) Or remote publishing to other peers in the subnet and Rendezvous. The receiving peers then save the advertisements in their cache. In the current implementation, we adopted the first possibility.

Table 3. Advertising methods

Methods	**Description**
PublishGroup(P2PNetwork group, P2PNetworkAdvertisement advert)	Publish the peer group advertisement
publishChannel(P2PNetwork group, ChannelAdvertisement advert)	Publish the channel advertisement
publishContent(P2Pnetwork group, ContentAdvertisement advert, boolean remote)	Publish the content advertisement

– **Discovery**. Methods offered by our platform allow to search group, channel and content advertisements. The discovery depends also on the underlying P2P layer. JXTA supports the following two main discovery mechanisms: (i) LAN-based discovery. Discovery queries are broadcasted over the network subnet. It corresponds to a fully decentralized P2P topology. (ii) Discovery via Rendezvous peers. It corresponds to a centralized

P2P topology. The current implementation adopts the first possibility for discovery. To avoid waiting indefinitely, discovery is limited by *timeout*.

Table 4. Discovery methods

Methods	Description
searchGroup(P2PNetwork group, String groupname, int timeout)	Search a peer group advertisement
searchChannel(P2PNetwork group, String channelname, int timeout)	Search a channel advertisement
searchContent(P2PNetwork group, String searchcriteria, boolean onlyone, int timeout)	Search a content advertisement

3.4.3. Requester module. This module provides facilities to establish direct communication with requested peer in order to retrieve a content and its descriptor. The API consists of the follwing key parts:

– *ConnectionRequester* is used to initiate a connection to another peer. Method *connect()* is used for this purpose. Once the connection is established, method *sendQuery(Message)* sends the given message to the requested peer. The returned response message is received through the *receiveResponse()*.
– *RequesterAccess* abstracts a communication endpoint in the requesting node side. Using JXTA as the underlying P2P substrate, *RequesterAccess* wraps the *JxtaSocket* API.
– *DownloadListener*. Listener interface used to customize the treatment of the retrieved object when a message arrives.

3.4.4. Provider module. This API gives the possibility to setup servers waiting requests coming from others nodes. It consists of:

– *ConnectionThread* that waits for connections from others nodes. It must be linked to one or more local repositories that stores *Content* objects. *ConnectionThread* can accept multiple connections from multiple peers and creates seperate communication session for each one. In the current implementation, it abstracts the *JxtaServerSocket* API provided by JXTA.
– *ConnectionProvider*. For each download query message *ConnectionThread* creates a new instance of *ConnectionProvider*. This allows to answer many queries at the same time.
– *ProviderAccess* abstracts a communication endpoint in the requested node side. It wraps the *JxtaSocket* API.
– *ConnectionListener*. The received query message is parsed in the method *receiveMessage(Message)*. This method returns another message containing a *Content* object or the content itself, to send to the requesting peer with *processResponse(Message)* method.

Each download step described in Figure 1 requires the following operations:

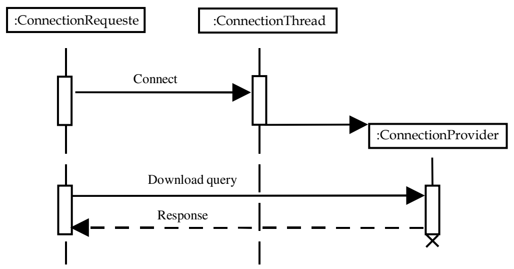

Figure 3. Detailed download step

The requesting peer and the requested peer exchange messages between them. Each message encapsulates two elements: the head and the body.

Table 5. Message encoding

Head	Body	Description
000	Content id	Content identificator of the content descriptor requested
100	*Content* object	The requested content object that contains the id and the descriptor of a content.
001	Content Id + content file path	Content Id of the requested content with the file path
101	Content data	Content bytes that represent the requested content.
111	Error message	Contains a message error indicating the requested object could not be found

3.5. Application layer

Applications layered on top of our platform such as service provisioning systems, must export the following method:

– *processMessage(Message)* called by a requesting node when a message response is received. It is possible to customize the treatment of the transmitted object after download by coding instructions within this method.

4. Performance study

The performance of the platform implementation was evaluated using simple but realistic scenarios. These scenarios investigate the impact of different parameters on system performance. Experiments are divided into two parts:

– The first part focuses on content discovery and provides results about the JXTA discovery service.
– The second part focuses on content delivery and gives results concerning the communication capability offered by JXTA.

For each part, tests are executed according to two scenarios:

– Local. Peers involved in the experiments run on the same computer independently from the physical network.
– LAN. Peers run on separate computers connected through a 100Mbps Ethernet Switch.

We have performed a large number of experiments. However, we report only results which help us to understand the system behavior. These results are organized in the follwing way. At first we report results related to the content discovery. Then we move to the content delivery performance evaluation. At the end of each set, we discuss the results.

4.1. Content discovery evaluation

In the local configuration, two peers run on the same PC, while in the LAN configuration peers run on different PCs. Every peer participates in the discovery test by searching and sharing contents with others in the P2P network.

Each peer provides n differents content advertisements, thus there are (n * group size) different content advertisements in the network. Each peer periodically broadcasts a discovery query message for a specific content chosen at random and records the response message when arrived.

For each query we measured the delay called the response time or round-trip time perceived by the peer. It includes the time for transmitting the query to the network, locating the desired content advertisement, and returning a response back to the requesting peer. This metric gives the efficiency of the discovery.

Java provides s*ystem.currentTimeMillis()* method to measure the time. The precision of this method is about 16 ms under Windows. This approach cannot give a good precision of measurement since operations that take less than 16 ms will appear to take the same amount of time. To solve this problem, we used the Java Native Interface (JNI) to access the high-resolution performance counter in Windows.

To evaluate the discovery we analyze the impact of the following factors on performance: amount of queries sent by each peer, delay between queries, advertisement cache size, and peer group size (only for LAN configuration). In the following, we present the results obtained including their analysis.

4.1.1. Local configuration. We varied the number of discovery queries sent by each peer from 100, 200, up to 1000, and from 1000, 2000, up to 5000. At the same time we maintained a fixed interval between queries (1000 ms). The number of advertisements per node was also fixed (100 adverts). We selected some experiments and reported the results for the response time on Figures 4, 5 and 6.

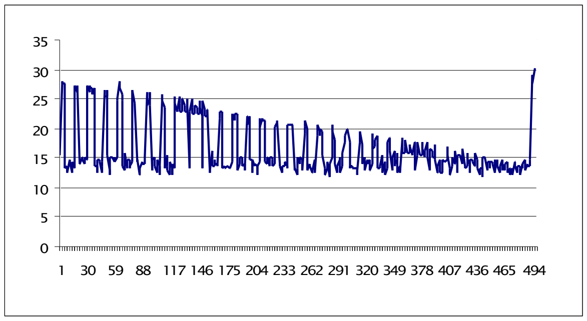

Figure 4. Content discovery response time for 500 queries

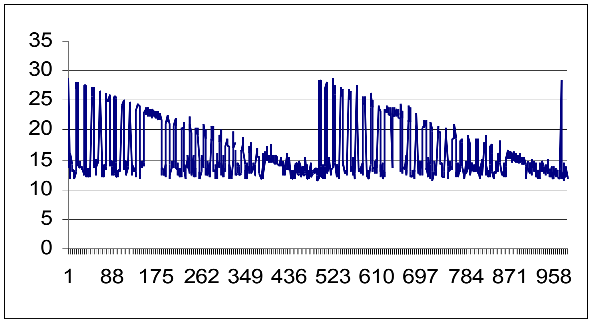

Figure 5. Content discovery response time for 1000 queries

Curiously, the obtained figures show the same behavior repeating every 480 queries. The time consumed in each period is 480 * 1000 / 60 = 8 mn for all tests. We also note that the average maximum response time value is about 28 ms while the average minimun response time value is 11.5 ms.

In the next scenario, we varied the number of published content advertisements by each peer to 1000 adverts and we repeated the same tests. The query rate is also maintained at 1 q/s.

Results show similar behaviors with the previous study. The time consumed in each period is also 8mn for all tests.

We note that the average maximum response time value is about 30ms while the average minimun response time value is 13.5ms. As expect, the variation of cache size from 100 adverts up to 1000 adverts introduces an overhead (2ms). This is a very slow increase compared to the number of published content advertisements.

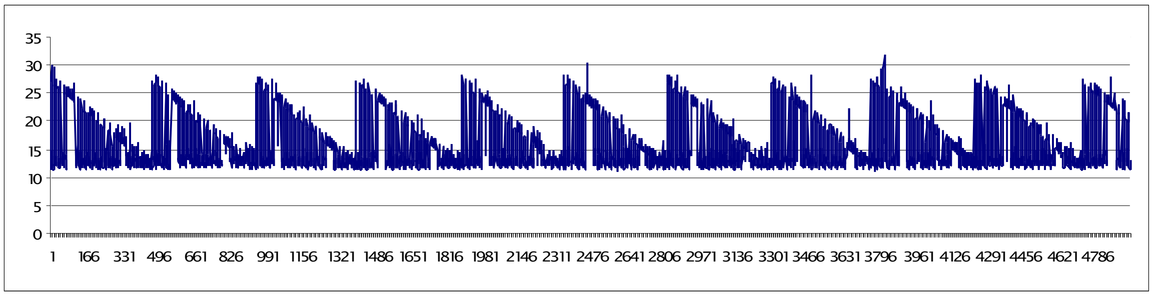

Figure 6. Content discovery response time for 5000 queries

In the next experiment, we fixed the number of queries at 1000, the cach size at 100 adverts and varied the query rate from 1, 2, 4, 8, 16, 32, up to 64 q/s. We ran each schema several times. Figures 7, 8, 9 and 10 report some selected experiments.

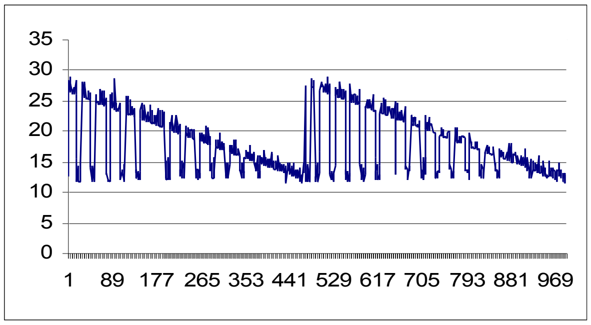

Figure 7. Content discovery response time – 1 q/s

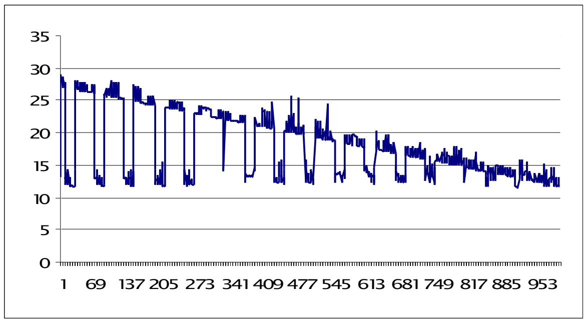

Figure 8. Content discovery response time – 2 q/s

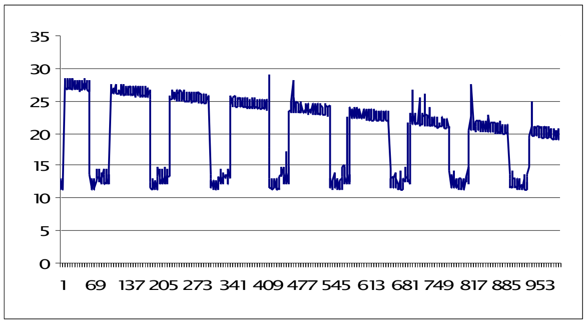

Figure 9. Content discovery response time – 4 q/s

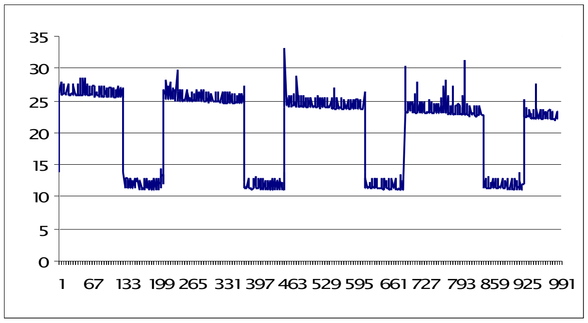

Figure 10. Content discovery response time – 8 q/s

Our results showed that, regardless of the delay between queries, the time consumed for each repetition period is always about 8mn. There is another behavior reported from tests with high query rate. It concerns the response time variation within each period. We can see that this time tends to be at high value for 20s then at low value for 10s for all tests. This behavior is also periodically repeated.

For very high query rate (32 and 64 q/s), the response time doesn't follow a unifom distribution and it is too slow compared to the other results.

We shall defer the discussion about these results after reporting LAN performance evaluation.

4.1.2. LAN configuration. The experiment environment consists of 8 PCs with Intel Pentium 4 2.80GHz CPU and 1.00GB of RAM. The operating system was Windows XP Professionel for all machines. In order to have relevant performance results these PCs run on a fully dedicated network. The formed P2P overlay follows a decentralized topology referred to as *direct communication architecture*, in which all peers can communicate directly with each other [8].

We conducted several experiments to see the effect of the network load over the discovery performance. Each peer stores 100 different content advertisements (artificially generated). We varied the number of discovery queries sent by each peer as follows: 100, 500, 1000, 5000, and 10000. For each set we varied the query rate from 1, 2, 4, 8, 16, 32, up to 64 q/s. We ran each schema several times.

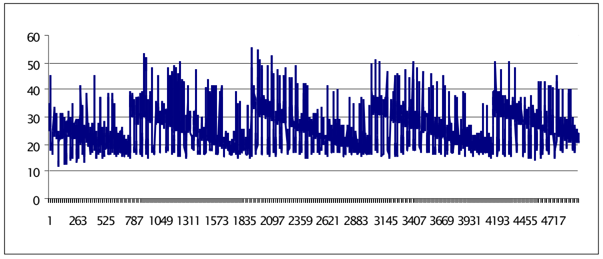

Figure 11. Content discovery response time - 2 q/s

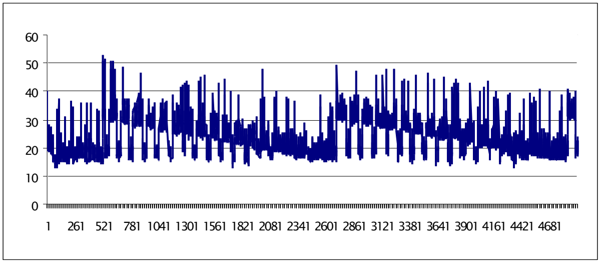

Figure 12. Content discovery response time – 4 q/s

From these tests, we obsereved a periodic behavior occured approximately every 8mn. Within each period we also oberved another periodic behavior as reported from the local configuration. Response time remains also at high level for 20s then at low level for 10s.

We denoted that the average maximum response time value is about 50ms while the average minimun response time value is about 16ms. Indeed the LAN introduces an overhead to the response time value.

We are also surprised about the number of discovery response messages received by each requesting peer. For each query the network returns up to 6 responses. We recorded only the first message.

4.1.3. Discussion. The experiments showed interested findings. In this section we provide insights into the reasons behind the observed behaviors.

In our approach we supposed that every peer published 100 different advertisements in its local cache, so it is expected that the returned response for each query is only one message containing only one advertisement. Moreover the JXTA documentation mentioned that if a peer has several requested advertisements, it returns only one discovery respone message containing all matched advertisements.

In consequence we can affirm that more than one peer participate in the discovery response. But how did these peers obtain the requested advertisements ?

Learning more deeply the JXTA specification we found that the JXTA discovery service automatically and periodically disseminates the local advertisements in the peer group every 30s. This value corresponds to the period time observed in the experiments. The dissemination requires CPU and memory ressources. That's why we observed a response time at high level for 20s then a response time at low level for 10s. We concluded that the dissemination step requires 20s to advertise 100 content advertisements.

In order to verify this assumption we extracted the expiration time included in the discovered content advertisements. We found 2 hours. We concluded that the periodic behavior is really due to the dissemination process because in JXTA, the default expirations for locally created advertisements is one year and 2 hours when advertisements are remotely published.

We also observed a periodic bahavior every 8mn. It means that there is another periodic task that occurs every 8mn and requires CPU and memory ressources. We think that this task may correspond to the garbage collector process since our implementation uses the Java language.

As expected, the number of advertisements and the communication over a LAN affect the response time and introduce an overhead. But this overhead is so small and doesn't perturbe the efficiency of the system.

4.2. Content delivery evaluation

4.2.1. Experiment setup and results. The goal here is to assess the performance of the content delivery feature offered by our platform. Experiments involve two peers: a requesting peer and a requested peer. In the local configuration, peers run on a single PC while in the LAN configuration peers run on two PCs connected through the network infrastructure.

For the requesting peer, we used the APIs exported by the requester module to implement a Java-based program that sends periodically (every 1000 ms) a download query to a requested peer and waits for the reply before sending the next query. The query message corresponds to the first row of Table 5. It contains a content Id chosen at random.

For the requested peer, we also used the APIs exported by the provider module to implement a Java-based program that waits for incoming queries. For each received query the requested peer parses its local repository and returns a response message containg the requested *content* object to the requesting peer which in turn records the arrival time.

We repeated the experiments many times and varied the *content* object size from 16 KB, 32 KB up to 64 KB. The chart below plots the average response time.

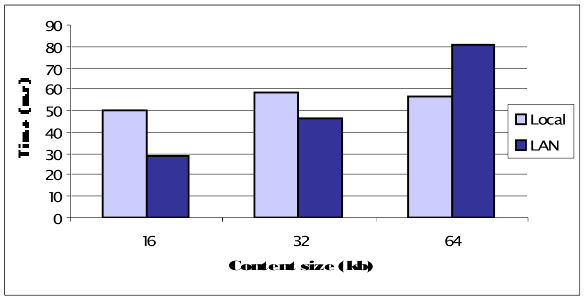

Figure 13. Content delivery performance

Concerning the local configuration, we obtained 50.39 ms for 16 KB *content* object size, 58.86 ms for 32 KB and 56.30 ms for 64 KB as average response time values. This time rises from 28.38 ms for 16 KB, 46.44 ms for 32 KB, up to 81.06 ms for 64 KB in case of the LAN configuration.

4.2.1. Discussion. In the rest of this section we discuss the results of the experiments.

First, in terms of functionality, the content delivery stage has achieved its purpose: a requesting peer obtains a *content* object from a requested peer after sending a download query message.

Second, the data collected show very good performance since the transfer delays are so small and the queries are successfully resolved. This denotes the efficiency and the effectiveness of the proposed system.

Lastly, Figure 13 shows also a linear dependence of the response time on the *content* object size in case of the LAN configuration. Curiously this linear dependence is not observed in case of the local configuration. As the graph shows the variation in response time is not significante. Moreover times are superiors to the response times in case of the LAN configuration (except for 64 KB). This behavior may be attributed to the fact that the two peers are executed on the same PC. The requested peer generates high loads (CPU and memory) because it parses all the content set in order to find the requested object. That's why response times are superiors. However the transfer occurs locally so there isn't a significant variations of the results. In case of the LAN configuration the preparation of messages to send and the access to the physical network layer requires an additional time that depends on the data size.

5. Related works

As we proposed a generic framework for content distribution, we focus the related work on this direction.

WiredReach [9] offers a framework to share any kind of content. It is also based on peer-to-peer model where a node acts both as client and server of contents. The main difference with our proposition is that WiredReach follows a centralized approach. Moreover nodes are first invited to be logged in the sharable space. In contrast our proposition is more general since it isn't limited to the centralized approach. Nodes can join and leave the shared space without invitation.

JXTA CMS [10] is a service built on top of JXTA for sharing and discovery of files. These files are described with advertisements published in the peer-to-peer network. In our opinion its main disadvantage results from the fact that all operations like advertising, discovery and retrieval are grouped in the same class CMS. This does not provide a modular solution for content distribution. [11] improves the JXTA CMS capabilities by adding a new layer that supports the discovery based on meta-data. However class CMS remains unchanged.

In [12], authors propose asynchronous dissemination of information over peer-to-peer network. Thanks to presented concepts, it is possible to set up a cooperative service for document exchange. They focus mainly on ad hoc networks. Providers broadcast periodically the same documents, which are discovered by other nodes at regular intervals. In our approach, the provider module can be used to publish content advertisement only one time or periodically depending on the application level. Requester module allows discovering contents at any time.

Edutella [13] is an open source project that creates an infrastructure for sharing metadata in RDF format. It is also based on JXTA. The main difference with our approach is that Edutella offers a specialized query service to deal with RDF metadata stored in distributed repositories. This method presents several advantages but it can have high complexity and therefore operate slow.

In terms of JXTA performance, [14], [15] and [16] analyzed the JXTA performance through several experiments. It is out of scope this paper to detail each research. However our main remark concerns the measurement methodology. Most of them use the *System.currentTimeMillis()* method. This method gives poor results compared to our approach.

6. Conclusion and future works

In this paper the synergy between Peer-to-Peer and service provisioning paradigm was recognized. First, we have identified the life-cycle of a service including: advertisement, discovery and delivery steps. Then we have studied and realized a P2P-based platform that supports these operations in a generic way. We have described the architecture of our framework and detailed the exported modules. Our main contribution focuses on the middle layer, which provides generic APIs used to build advanced P2P applications. The term generic is justified by the fact that these APIs are independent from the context of use. Separation between layers and the different operations provides modular architecture, which can be adapted to various fields, like the service provisioning field. Finally we analyzed a large number of experiments on different scenarios. The experimental results are divided into two parts: experiments to test content discovery and experiments to test content delivery. Results have shown the efficiency and the effectiveness of our approach.

Future works can be pursued according to three main directions:

– Implementation of the API in the context of wireless devices like PDAs and smartphones,
– Analyzing other metrics like data throughput to improve the performance study,
– Exploring advanced discovery techniques based on semantics and context to find relevant contents.

7. References

[1] J. O'Sullivan, D. Edmond and A.H.M. Ter Hofstede, "What's in a service? Towards accurate description of non-functional service properties", *Distributed and Parallel Databases*, 12, 2002, pp. 117-133.

[2] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies", *ACM Computing Surveys*, vol° 36, Issue 4, December 2004, pp. 335-371.

[3] K. Aberer and M. Hauswirth, "An Overview on Peer-to-Peer Information Systems", *Proceedings of Workshop on Distributed Data and Structures (WDAS-2002)*, Paris, France, 2002.

[4] M. Ehrig and al., "Towards Evaluation of Peer-to-Peer-based Distributed Information Management Systems", *Agent-mediated Knowledge Management (AMKM 2003), AAAI Spring Symposium 2003*, Stanford, March 24-26, 2003, pp. 73-88.

[5] F. Benayoune, L. Lancieri, "Models of Cooperation in Peer-to-Peer Networks, A Survey", *3rd European Conference on Universal Multiservice Networks (ECUMN 2004)*, Porto, Portugal, October 25-27, 2004.

[6] S. Frénot and Y. Royon, "Component deployment using a peer-to-peer overlay", *Working Conference on Component Deployment*, Grenoble, France, 28th-29th November 2005.

[7] L. Gong, "JXTA: A Network Programming Environment", *IEEE Internet Computing*, May 2001, p. 88-95.

[8] J. Walkerdine, L. Melville, I. Sommerville, "Dependability within Peer-to-Peer Systems", *In the proceedings of the ICSE Workshop on Architecting Dependable Systems (WADS) 2004*, Edinburgh, 25 May, 2004.

[9] WiredReach, "Open source development framework for building and deploying user centric web applications", http://www.wiredreach.com

[10] CMS, "The JXTA Content Manager Service", http://cms.jxta.org

[11] X. Xiang, Y. Shi, and L. Guo, "Rich Metadata Searches Using the JXTA Content Manager Service", *18th International Conference on Advanced Information Networking and Applications (AINA'04)*, vol° 1, 2004, pp.624.

[12] H. Roussain and F. Guidec, "A Peer-to-Peer Approach to Asynchronous Data Dissemination in Ad Hoc Networks", *International Conference on Pervasive Computing and Communications (PCC'04)*, Las Vegas, Nevada, June 2004, pp. 799-805.

[13] W. Nejdl and al., "Edutella: A P2P Networking Infrastructure Based on RDF", *The Eleventh International World Wide Web Conference (WWW2002)*, Honolulu, Hawaii, USA, May 7-11, 2002.

[14] J. M. Seigneur, G. Biegel, and C. D. Jensen, "P2P with JXTA-Java pipes", *Proceedings of the 2nd international conference on Principles and practice of programming in Java*, 2003

[15] E. Halepovic and R. Deters, "JXTA Performance Model", *Future Generation Computer Systems, Special issue on Peer-to-Peer Computing and Interaction with Grids*, Elsevier, vol° 21/3, 2004, pp 377-390.

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