December 2003

dAMP: A JXTA-Based Peer-to-Peer Media Player

Qusay Mahmoud
University of Guelph

Leslie Yu
University of Guelph

Christopher Long
Sierra Systems

dAMP: A JXTA-Based Peer-to-Peer Media Player

Qusay H. Mahmoud
University of Guelph
qmahmoud@cis.uoguelph.ca

Leslie Yu
University of Guelph
lyu01@uoguelph.ca

Christopher Long
Sierra Systems
chrislong@sierrasystems.com

Abstract

Traditional approaches for searching through websites for media files to download, and tracing broken links is a frustrating experience and a time consuming task for end users. This experience can be improved through Peer-to-Peer (P2P) computing. The notion of P2P or cooperative computing and resource sharing has been around for quite some time, but it wasn't utilized until file sharing solutions such as Napster and Gnutella emerged at the forefront of Internet services and opened up possibilities for flexible Web-based file sharing. These file sharing solutions, however, have their own limitations. For example, they still require a lot of manual management by the user. This paper describes the design and implementation of a distributed P2P media player called dAMP that brings about novel ways of discovering, downloading, and playing media files. The dAMP media player provides location transparency by enabling users to share and play media files as though the files are local.

Keywords: Peer-to-peer, P2P media player, P2P media streaming, JXTA

Introduction and Motivation

In sharp contrast to the traditional client-server architecture of the Internet, Peer-to-Peer (P2P) creates a completely new form of mutual resource sharing over the Internet (Rodrigues et al. 2002). With the increasingly common place broadband Internet access, P2P technology has finally become a viable way to share files. To download music files, users currently have two main options: to download off websites or to download off another user using a P2P client.

Downloading off websites is the traditional way of augmenting one's music collection. This method has been around since the beginning of the Web. However, there are many problems associated with this approach. The most obvious problem is broken links. Music files are usually hosted on websites run by daredevils not afraid of being sued by big record companies. These sites come and go as their webmasters try to evade the authorities. Another problem associated with this method is the generally chaotic manner in which most sites order their files. Last but not least, downloading from websites generally mean wading through screen after screen of pop-up advertisements! In short, downloading from websites is inefficient and time consuming.

On the P2P side, there are already programs on the market such as Napster (Napster, 1999) and Gnutella (Gnutella, 2003) that enable P2P file sharing. These programs enable millions of users to share files amongst themselves. While the utilization of P2P clients is already a gigantic step forward compared to downloading files off websites, using such programs are not without their problems. The downloaded files still require a lot of manual management by the user. The user still needs to put the files in the proper directory, categorize the files according to his preferences, import the files to the player and further, delete the files when they are no longer wanted. We strive to make the whole process easier.

Another problem with the two approaches mentioned earlier is that the files are duplicated all over the Internet. With the increasing commonality of home networks and the desire to collect huge libraries of music, users are forced to either divide their music among multiple computers or make a decision about which ones to keep and which ones to duplicate on multiple machines. Huge amounts of hard drive space can be potentially freed, if users get used to the idea of counting on someone else to house songs that they do not want a local copy of.

With its small size and wide popularity, MP3 music files are fast becoming the de-facto file format amongst music lovers in sharing their files. Coupled with the ease of file sharing over P2P networking technology, we have designed and implemented a novel system that integrates these two functions. With this combined functionality, dAMP minimizes the time and effort for the user to enjoy their favorite music.

The dAMP Solution

The old process of acquiring and then playing back of songs involve searching for songs, downloading them, importing to the media player, and perhaps deleting them afterwards. This is a very time-consuming task. Our media player, dAMP, strives to do better. dAMP shortens this cycle to just *search and play*. All other aspects are automated by the application and the user workload is dramatically decreased.

The dAMP player automatically shares the users' songs and starts playing what the user likes without any human intervention. First, the user specifies the type of music he or she likes to hear, and the program will take care of the searching, downloading, playback, and the subsequent file removal. We have designed this system to be used in a Local Area Network (LAN) setting, because of the fast connection. On a 100 Mbps LAN, it would take less than 1 second to download a 4-megabyte mp3 song. Files are downloaded only when they are needed and they are automatically deleted after they are played.

The dAMP player acts as a virtual hard drive, containing media files of all the clients on the network. This will eliminate the duplicated files problem. Every user will specify a folder containing media files that they will be sharing with other users. In true decentralized fashion, the client is P2P, and will not need a central server to serve the files. The application uses XML to represent the list of songs that are in the peer group. This allows the application to be platform independent and works over heterogeneous networks and platforms. Users will not encounter the broken link problem because the play list is constantly updated and contains only the currently available songs in the group. In a school or office LAN environment where computers are always on, sharing of files is always available.

The dAMP Design

The dAMP media player is designed to provide location transparency. In other word, users will not notice the files are local or on a peer's system. We hide the information of which files belong to which peer. In addition, there is no difference in playing a local or a remote file except for a very small latency if the file has not been cached. We designed the application to operate in a heterogeneous platform and network environment by using platform independent technologies such as Java, XML, and JXTA (Gong 2001), which is a P2P Java-based framework.

The reason we chose JXTA as our main distributed model rather than the traditional client-server model is because it provides us with scalability, direct user-to-user communication and up-to-the-minute searches. Although the bandwidth of the server might be very high and more bandwidth might be added. As the number of users increase, the amount of bandwidth that each user gets is very limited if many users access the server at the same time. JXTA solves this problem by having the file transfer between peers, not between client and server. The files on the server will be unavailable if the server is down, JXTA allows the direct transfer between peers so that only the files of the crashed peers will be unavailable, others will still be up. Search engines may contain tons of web pages for the users to search, but the information may be out-of-date since indexing does not happen all the time. One of the major features in our system is to search peers for songs. JXTA provides us with always up-to-date information and only available files will be returned instead of broken links.

The dAMP Architecture

The dAMP architecture consists of three layers: Presentation Layer, Logic Layer, and Transport Layer as shown in Figure 1. The Presentation Layer has two modules: the GUI module and the Player module. It contains the logic to handle the user commands and playback of the media files. The Logic Layer has two modules: the Search module and the Media Handling module. This layer

lists the songs available and the download and deletion of files. The Transport Layer has one module; Peer-to-Peer. This layer sets up the connection to the peer group and handles the communication between peers.

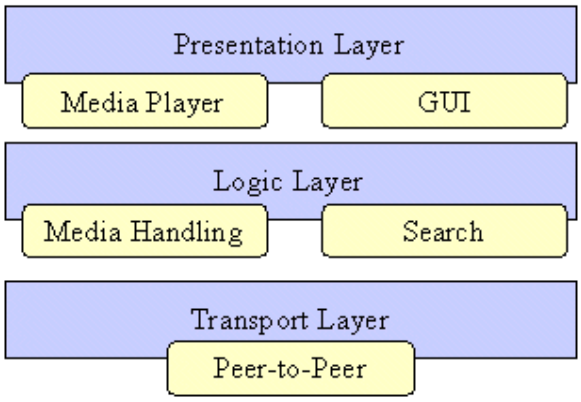

Figure 1. The dAMP Architecture

GUI Module

This top-level module provides the graphical user interface (GUI) to the end user for controlling various aspects of the entire application. For example, the user can update the pre-set preferences on what kind of music they would like to hear, and they can control the playback of the music files. The event messages generated by the user's interaction with the controls are translated, and function calls are made to the various modules to carry out the task. This abstracts away the complexity of the underlying peer-to-peer architecture, and hides the user from the complicated issues of network communication faults, multi-threading, etc.

Media Player Module

This module takes a media file and plays it. The order of files that it plays is decided by the playlist generated by the Search Module, and the user can control the playing of the files by interacting with various controls in the GUI, which is controlled by the GUI Module.

Search Module

This module provides many of the core functions of the program. Its primary task is to maintain a playlist of songs that matches the user's pre-set preferences of music, and preparing them to be played by the Media Player Module. This is accomplished by:

- Making use of the services provided by the Peer-to-Peer Module to broadcast an XML song request message to all the peers. The content of this message reflects the user's searching criteria set in the GUI module. This broadcast is done at regular intervals (such as once a minute) to ensure that the local playlist provides an up to date portrayal of the availability of songs.

- On the side of the remote peer, once a request message is received, it searches through its local file list and compiles a reply XML message containing a list of songs that match the search criteria. This message is sent back privately to the requesting peer.

- The requesting peer receives the XML message containing the playlist and merges all those entries in the local playlist. In effect, the local peer's playlist has just been augmented with songs that are hosted remotely. For entries that are already in the playlist, we consider that reply as a "pulse". A constant inflow of pulses from a remote peer indicates those

files hosted by said peer are in good health. When we do not receive any pulse from a remote peer within a timeout period, we consider those files dead. They will be removed from the playlist.

An interesting twist to our implementation of the search module is that the local machine is treated just like any other peer. When the initial request is broadcasted, the local machine receives that request just like any other peer, and responds to it just like any other peer. This embodies the idea of location transparency where the server code does not know, nor care where the request is coming from.

In addition, it also keeps track of a local file list, containing information on all the shared files in the shared folder. When the application is started, this module imports all the MP3 files from the shared folder to the local file list. At the same time, it reads the ID3 tags of each of the MP3 files to determine information such as artist, song title, media file length, bitrate, and other information, which can be used as the search criteria.

Media Handling Module

Using the services of the Peer-to-Peer Module, it downloads the files in the playlist sequentially in the background. It is also responsible for cleaning up songs in the cache that have already expired.

Peer-to-Peer Module

This module is responsible for handling the lower level peer-to-peer communications. It carries out the task of logging into the JXTA networking and joining the proper peer-group. It sends and receives messages between peers asynchronously for the co-ordination of searching and the actual file downloading. It provides services for modules in a higher level.

The dAMP Protocol

We have designed and implemented a special protocol for dAMP, which allows peers to communicate effectively, as shown in Figure 2. All messages sent between peers are encoded in the protocol format, which in turn defines the XML messages. The Data Type Definition (DTD) of the XML-based dAMP protocol is given in Appendix A, and an example of a message exchange of the protocol is given in Appendix B.

The dAMP Implementation

The system was developed using the Java 2 Standard Edition Development Kit (J2SE 1.4). We have used the NetBeans IDE, which provides tools such as drag-and-drop GUI development that automatically generates much of the Java Swing code for the GUI. We implemented the player using the Java Media Framework API to handle MP3 playback. The system extracts song information inside the ID3 tags using the JD3Lib API (Hilliker, 2001). The information is then encoded in XML format to allow the peers to search for songs using the dAMP protocol. The underlying P2P layer is implemented using the JXTA protocol. All the tools used are platform independent, which allows the system to be run in a heterogeneous network. A screen shot of the dAMP user interface is shown in Figure 3.

Play List Reply Message:

Clip	Clip	Clip	Clip	Clip	Clip	Clip	Clip	Clip

Clip Parameters:

Title	Artist	Genre	Type	File Name	File Size	Bitrate	User	Length

Play List Request Message:

Category	Search String	Category	Search String	....	Category	Search String

File Request Message:

File Name

File Reply Message:

File Name	Status	Binary data

Figure 2. The dAMP Protocol

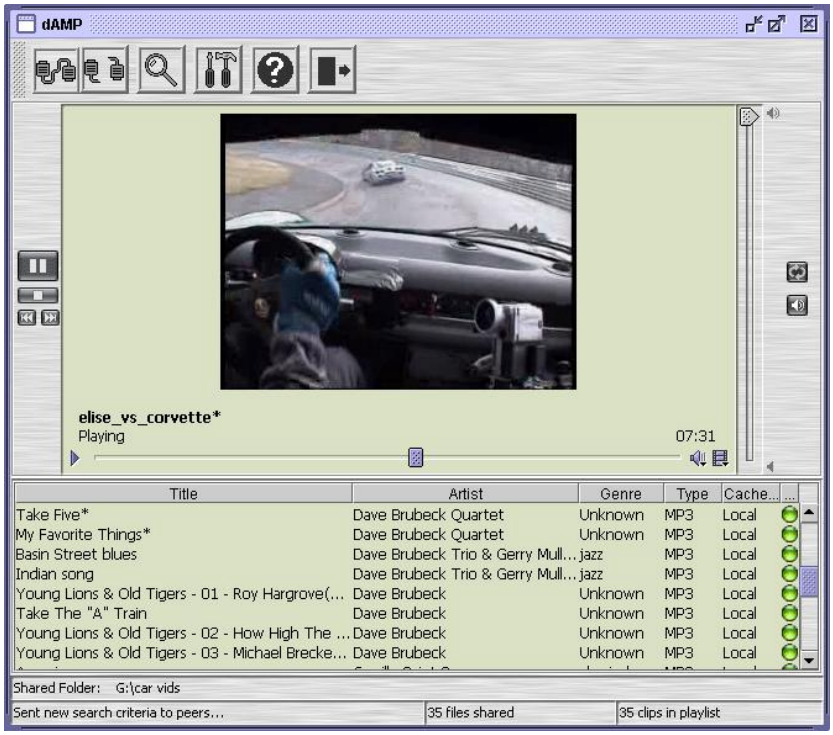

Figure 3. Sample Screen Shot of dAMP (Playing a Video File)

Design and Implementation Challenges

We have met the following challenges during the design and implementation of the dAMP system:

- Working with an asynchronous cross-platform P2P architecture is challenging. P2P is a complex paradigm with many concepts and concerns.
- There is a large amount of messages that gets passed internally and externally. Internal messages are implemented via callbacks, while external ones are done over the dAMP protocol.
- JAXTA is still under development. There are still significant bugs. The technology is not very well documented for an API of its size. Many of the sample programs supplied by the JXTA release do not work well. Much effort was spent figuring out why JXTA did not work as advertised. Eventually a workable solution was developed using JXTA, but not without sweat.

Future Work

Currently, the application retrieves the files as the user requests to play a particular song. It is fast on a Local Area Network (LAN) environment, but slow over the Internet. We plan to stream the files instead of first downloading the entire file and then caching it.

We plan to port our implementation to the Java 2 Micro Edition (J2ME) so that it would work on Java-enabled mobile devices such as cellular telephones and personal digital assistants. Before we do so, however, we'd like to address the security and privacy concerns.

Security issues need to be looked at more closely. The dAMP system, being a P2P client, not only accepts requests from anyone, but also makes its own shared folder world readable. We are not very concerned about message masquerading because the folder is already world readable. Hence, there is no need to pretend to be someone else to access the folder. We are more concerned about denial of service attacks. Currently, we do not protect ourselves against that. A future enhancement could be to throttle message traffic from users to prevent flooding of the channel. Another attack we need to guard against is remote users gaining access to folders other than that of the shared folder. This possibility of jeopardizing the security of the entire system is a grave concern for us. Currently, we make sure no relative paths are used in file requests.

Finally, we have already started to extend the system to handle other types of media files such as mpeg, avi, and others.

Conclusion

The P2P revolution has just started and exciting possibilities are opening up this technology. File sharing solutions such as Napster and Gnutella have their own limitations. For example, they still require a lot of manual management by the user. In this paper, we described the design and implementation of a distributed P2P media player called dAMP that brings about novel ways of discovering, downloading, and playing media files. The dAMP media player provides location transparency by enabling users to share and play media files as though the files are local.

In the design and implementation of dAMP, we came to appreciate the complexity and the issues involved in building a successful distributed P2P application. We come to realize the difficulties of ensuring the security of a system while making one part of it world readable. Security and privacy concerns are major issues in P2P systems and we'd like to explore this area in dAMP.

References

Gnutella: **http://www.gnutella.com**, 2003.
Gong, Li.: JXTA: A Network Programming Environment. IEEE Internet Computing, May-June 2001. pp 88 – 95.
Java 2 Micro Edition (J2ME): http://java.sun.com/j2me.
Hilliker, J.: JD3Lib: http://sourceforge.net/projects/jd3lib, 2001.
Napster: http://www.napster.com, 1999.
Rodrigues, R., Liskov, B., Shrira, L.: The Design of a Robust Peer-to-Peer System. Available online: **http://www.pmg.lcs.mit.edu/~rodrigo/ew02-robust.pdf**.

Appendix A. DTD Associated with the dAMP Protocol

```
<!ELEMENT damp_message (playlist_reply*,playlist_request*,file_request*,file_reply*)>
<!ELEMENT playlist_reply (playlist)>
    <!ELEMENT playlist (clip*)>
         <!ELEEMNT clip (title, artist, genre, type, filename, filesize, bitrate, user,
                         length)>
             <!ELEMENT title (#PCDATA)>
             <!ELEMENT artist (#PCDATA)>
             <!ELEMENT genre (#PCDATA)>
             <!ELEMENT type (#PCDATA)>
             <!ELEMENT filename (#PCDATA)>
             <!ELEMENT filesize (#PCDATA)>
             <!ELEMENT bitrate (#PCDATA)>
             <!ELEMENT user (#PCDATA)>
             <!ELEMENT length (#PCDATA)>
<!ELEMENT playlist_request (entry*)>
    <!ELEMENT entry (category, param)>
         <!ELEMENT category (#PCDATA)>
         <!ELEMENT param (#PCDATA)>
<!ELEMENT file_request(filename)>
<!ELEMENT file_reply (filename,status)>
    <!ELEMENT status (#PCDATA)>
```

Appendix B. An Example of the dAMP Protocol

```
<?xml version="1.0"?>
<!DOCTYPE dample_message SYSTEM "xml/dAMPProtocol.dtd">
<damp_message>
    <!-- Playlist reply -->
    <playlist_reply>
        <playlist>
            <clip>
                <title>One</title>
                <artist>U2</artist>
                <genre>rock</genre>
                <type>MP3</type>
                <filename>U2 - one.mp3</filename>
                <filesize>4500000</filesize>
                <bitrate>128</bitrate>
                <user>damptest</user>
                <length>342</length>
            </clip>
        </playlist>
    </playlist_reply>
    <!-- Playlist request -->
    <playlist_request>
        <entry>
            <category>Genre</category>
            <param>rock</param>
        </entry>
        <entry>
            <category>artist</category>
            <param>tragically</param>
        </entry>
    </playlist_request>
    <!-- file_request -->
    <file_request>
        <filename>U2 - one.mp3</filename>
    </file_request>
    <!-- File reply -->
    <file_reply>
        <filename>U2 - one.mp3</filename>
        <status>Not Found</status>
    </file_reply>
</damp_message>
```