Using Mobile Agents for Network Resource Discovery in Peer-to-Peer Networks

Cameron Ross Dunne
School of Computer Applications, Dublin City University, Dublin 9, Ireland.

Peer-to-Peer networks continue to grow in popularity. However network resource discovery still remains a substantial problem within them. In this paper we will cover some of the more popular current solutions to this problem. We will then propose a mobile agent based solution to allow for dynamic network resource discovery.

Keywords: Mobile Agents, Resource Discovery, Peer-to-Peer.

1. Introduction:

Peer-to-Peer, commonly known as P2P, is the movement away from the more traditional client-server model to a network where each participating device is acting as both client and server [1]. Some of the main advantages of P2P are greater storage, more computer cycles, and greater bandwidth. P2P technology is also gaining popularity due to the current economic slowdown, because users are looking for ways to get more use out of their existing hardware.

Most Internet users have encountered some of the more popular P2P systems to date, for example Napster [2], SETI@Home [3], or ICQ [4]. P2P is now starting to be used in many more application areas, including e-commerce. eMikola [5] are developing a product called "Peer Switches", that is used for more efficient information distribution, and is based on P2P technology. Another P2P based example is Groove [6], a product that allows users to create shared spaces over the Internet that can be used for collaborating projects, virtual meetings, or trading.

For P2P applications to succeed, a number of the underlying technologies must be developed. In the remainder of this paper, we will be focusing on the issue of efficiently and successfully locating network resources on a P2P network.

1.1. Software Agents:

Although there is no universally accepted definition of a software agent, most agree that it is a computing entity that performs some task or tasks on behalf of somebody or something [7]. There are many additional features that software agents may employ, such as possessing intelligence, or a user interface. However such features are not essential. When the software agent always operates within the system where it was started it is often known as a "stationary agent".

1.2. Mobile Agents:

A mobile agent is a software agent that has the additional property that it is not bound to operate only in the system in which it started. A mobile agent has the unique property that during its lifetime it can be halted, its state and code moved to another computer on the same network, and then continue executing from where it stopped executing on the previous computer. A mobile agent is autonomous because it may decide itself where it will go, what it will do there, and how long it will exist for. However, its environment or other mobile agents may also influence it.

Although mobile agents do not provide a solution to any previously unsolvable problems, they do have advantages over other technologies [8]. They can be used to benefit or to simplify different types of application areas. Some examples of these application areas include e-commerce, distributed information retrieval, telecommunication networks services, and monitoring and notification [9].

1.3. Aglets:

There are various different implementations of mobile agents in existence [10]. Some of the more popular implementations include Telescript (General Magic, Inc. [11]), and Aglets (originally IBM Japan [12]). In the remainder of this paper we will be focusing on the Aglets implementation of mobile agents.

The Aglets project is a Java [13] based implementation that was originally developed by IBM in Japan. However late in 2000 they made the project open source, and it is now developed and maintained by the Aglets community [14]. Some of the reasons for Java's suitability for developing Aglets are:

- The popularity of Java makes Aglet development quicker and cheaper.
- The portability of Java compiled code due to its "write once, run anywhere" architecture.
- The Java Virtual Machine has a built-in, fine-grained, and very configurable security control mechanism.
- Java has built-in support for network programming.

2. The Problem:

Currently all searching on the Internet and large networks is carried out by dedicated centralised search engines, such as Yahoo [15] or Google [16]. Developing such systems tends to be extremely expensive in terms of hardware, bandwidth requirements, and also the specialised algorithms and software that are necessary. Most Internet search engines maintain a very large centralised database, which is updated by crawling the Internet and indexing websites. When a query is received the database replies with a list of websites that are deemed to be in some way related to the original query.

So while current search systems may be suitable for general web searching they do have disadvantages. The current size of the Internet, and the rate at which is it growing, make it impossible to visit every website and index it. The most web pages that any search engine has indexed is 1.3×10^9, yet it is estimated that this is less than 20% of web pages in existence under the ".com" domain [17]. Another significant disadvantage is that indexing only works well on websites that contain static information, as the search engine may not visit the website regularly enough to be able to index new content.

These disadvantages, along with the general architecture of search engines, make them unsuitable for searching in large dynamic P2P networks. Also, a P2P network will only be as popular as the usefulness of the resources that can be located within it. Therefore being able to efficiently search for and successfully locate these resources is extremely important.

3. Current Solutions:

A variety of different P2P systems try to address the problem of network resource discovery in different ways. Below we will examine some of the more popular systems, and identify some of their strengths and weaknesses.

3.1. Napster:

Napster was perhaps the first P2P network to gain widespread recognition and use. It gained support due to its ease of use, its accuracy, and of course the large demand that had developed for MP3 music files. Napster itself is a closed application, but there is an open protocol known as OpenNap that is based on Napster [18]. We will refer to both of these collectively as Napster.

The architecture of Napster is that it maintains a central database on a server with details of available MP3 music files and their locations [19]. When a user wishes to download a file the database is queried. If a match is found the client is given the details of where the file is located. The client then makes a direct connection to the peer client that possesses the file, and downloads it. After a successful download the central database is updated with the client address where a new copy of the file is now located.

So Napster is extremely fast and accurate for locating the correct files due to the central database it maintains. However it is this centrally located database that has led to Napster's legal problems [20], and the rapid loss of users [21]. It is unlikely that any future P2P system would use a similar architecture that leaves it open to similar legal actions.

3.2. Gnutella:

Gnutella is an open, text based protocol that is designed to allow for the sharing of all file types over the Internet [22]. A computer participating in a Gnutella network is acting as both client and server, and hence is called a *servent*. There are many different client applications that support the Gnutella protocol such as BearShare [23], and Limewire [24].

A computer wishing to participate in a Gnutella network does so by contacting another Gnutella servent [25]. The acquisition of this other servent's address is not actually part of the Gnutella protocol. Once connected, servents seek out other servents by sending Pings, which may be responded to with a Pong. When a servent receives a Ping it forwards it to all the servents that it knows of, and sends the resulting Pongs back to the servent that initiated the ping. To search for a file a Query is sent to all known servents. If the servent has a matching file it responds with a Hit, otherwise it forwards the query on to all the servents that it knows of. Once the file has been found it is downloaded outside of the Gnutella network using the Hyper Text Transfer Protocol.

Perhaps the biggest problem with Gnutella is that each servent uses so much bandwidth for sending and receiving Ping/Pong and Query/Hit messages that are not related to it. This is critical considering that these messages grow exponentially. It is also believed that most users on the Gnutella network are not making any files available for sharing, thus making the network very inefficient [26]. Another problem is that each servent only ever sees a relatively small portion of the entire network.

To address some of these problems Clip2 developed a Reflector system [27]. The Reflector is a super peer located between a group of end servents and the rest of the Gnutella network. When servents connect to the reflector an index is created on the Reflector with all the details of shared files on the servent. The Reflector thus is able to shield the connected end servents from all Ping/Pong and Query/Hit messages while also providing servents with greater network coverage.

3.3. Morpheus/KaZaA:

Morpheus and KaZaA [28] are two new file sharing applications being developed by FastTrack [29], and based on the same closed protocol. Both are currently attracting quite an amount of interest. They appear to operate in a similar way to a Gnutella network with Reflectors. Peers are self-organising, meaning that when they detect that they have enough bandwidth, they can take on the responsibilities of acting in a way similar to the Reflectors.

3.4. Project Juxtaposition (JXTA):

Sun [30] has entered the Peer-To-Peer market with the recent launch of Project JXTA, an open community layered set of protocols [31]. JXTA differs from other P2P projects for a number of reasons [32]. The JXTA protocols are completely hardware and language independent. Therefore two JXTA peers may communicate with each other over Bluetooth without ever needing TCP/IP, unlike any of the previously mentioned systems.

It is envisaged that many peers will not be PC based, but that they will be either enterprise systems or consumer-orientated small, systems such as PDAs. For this reason the protocols have been kept as thin as possible. Also, each peer need not implement every protocol. Project JXTA provides a layer on top of which P2P applications and services can be built, just like TCP/IP does. Finally, all the protocols defined within JXTA are extensible, allowing application builders to use their own implementations. For example some application builders may require weak security, while others may use their own strong security.

While a complete description of JXTA is beyond the scope of this paper, there are some aspects worth noting [33]. JXTA uses a small subset of XML [34]. However, peers need not be aware of this. Indeed, they can use predefined XML messages alone. JXTA consists of six protocols, but the ones most relevant to this paper are the Peer Discovery Protocol, and the Peer Information Protocol.

The Peer Discovery Protocol (PDP) allows a peer to find other peers, groups, resources, or services. It does so by sending XML based messages to rendezvous peers in a manner similar to web crawling. JXTA allows custom discovery services to be developed and used. Therefore the PDP is a lowest common denominator discovery protocol.

The Peer Information Protocol (PIP) is used to determine the status and capabilities of a peer. The status of a peer is determined by sending a Ping message, and waiting for the corresponding reply. The capabilities of a peer are described using a string to name the property, and a string to represent its value. PIP provides read-only access to these properties. However, read-only access may be replaced to give read/write access.

As of now JXTA is still being developed, and there are no applications based on it in widespread use yet. Project JXTA seems to be very well designed, but only time will tell how well it will succeed.

4. The Proposed Mobile Agent Based Solution:

4.1. Why use Mobile Agents?

Mobile agents reduce the need for bandwidth. Very often peers using a distributed protocol establish a communication channel between themselves, and then perform multiple interactions over this channel. Each of these interactions generates network traffic. Mobile agents allow these interactions to be packaged together, and sent as a discrete piece of network traffic. This then allows all the interactions to take place locally. Mobile agents also encapsulate all the required data within themselves. Therefore when a mobile agent arrives on a computer it has all its data with it, and does not need to communicate with any other computers. In a conventional search protocol all the raw data travels over the network to be processed, even though only a subset of this data may be needed. In this scenario, mobile agents reduce the network traffic by moving the processing to the raw data, instead of moving the raw data to the processing. Finally, mobile agents can be very small in size, but can grow dynamically as they need to accommodate more data.

Mobile agents are asynchronous. Therefore when a mobile agent is dispatched there is no need to wait for it to return. Indeed the original peer does not even need to remain connected to the network while the mobile agents are out. The mobile agents can wait until the original peer is back on the network before attempting to return to it.

Mobile agents are autonomous. This particularly suits network discovery, because the mobile agent is learning about the network as it progresses through it. The mobile agent will visit peers that were unknown when it was originally dispatched. At each peer it can make decisions based on its history of visited peers and the current peer.

Information is being disseminated at every peer that the mobile agent visits. Every peer benefits from accepting a visiting mobile agent, because the mobile agent will have either new or more recent information about resources. Also, every mobile agent benefits from visiting a peer because it will learn of either new or updated resources. If the mobile agents do not contain any new information they may be destroyed. Accepting and hosting mobile agents requires the use of physical resources, such as memory and computer cycles. Should these become critically limited, it is easy for the peer to refuse further requests to accept mobile agents until more physical resources become available.

Mobile agents may easily be cloned and dispatched in different directions. This allows them to function in parallel. Although this causes more mobile agents to be active on the network, it does ensure that the network resource discovery is completed sooner, and therefore the mobile agents spend less time on the network.

A mobile agent based solution is very fault tolerant. Even if some of the mobile agents are destroyed, all surviving ones will have a positive impact. Indeed, the destroyed mobile agents will have benefited every peer up to the point where they were destroyed.

Finally, a mobile agent based solution can be combined with successful features from other P2P based systems to provide an improved final solution.

4.2. What is a Resource?

A resource is something that may have a perceived value to somebody or something. Each peer participating in the network may make some of its resources available to other peers. We have tried not to make any assumptions on what a resource may be.

There are certain mandatory resources that a peer must posses to allow for the most basic network resource discovery. It must know what type of peer it is. The peer must be able to maintain a directory with details of each known peer, its type, and its address. This directory must be updateable by visiting mobile agents.

There are many other additional resources that a peer may have, and these may or may not be updateable by visiting mobile agents. The resource may be a directory of files and their

locations known to the peer. These files might be a particular category, such as music, or movie clips. A complex software object that the visiting mobile agent can interact with may be the resource. The resource may be available computer cycles, along with details of when they are available. The resource may be some dedicated hardware, such as a network router or switch.

4.3. Peer Discovery Algorithm:

The main algorithm needed is used to decide how the mobile agents will behave in their search for other peers. The algorithm is not trying to perform total network discovery because this is not at all scalable to very large networks. So the algorithm is making a trade-off between a more detailed network discovery and a more efficient and practical network discovery.

After the mobile agents have completed their work, the original peer will have built a view of the part of the network that it now knows about. This view is essentially an undirected graph [35]. The mobile agents have performed a basic Breadth-First search in parallel with each other.

The algorithm adopted by us is not necessarily the best possible algorithm, as the algorithm was not the focus of our work. We recognise that graph theory is an area of research in its own right. Indeed, work has already been performed on viewing the World Wide Web as a graph, and designing suitable searching algorithms based on this view [36].

The algorithm we used is as follows:

1. A mobile agent is created on a peer wishing to participate in the network. This peer is known as the "creator peer" for that particular mobile agent, and all future clones of it. The mobile agent updates itself with information about its creator peer, and in particular notes the address of this peer as its home address. A journey-time for the mobile agent is set in terms of the maximum number of peers that may be visited before it must return to its creator peer. Also a branching factor is set, that is used to determine how many times the mobile agent may be cloned at any one peer. These two parameters control the depth and breadth of the search, and therefore the maximum number of peers that may be visited.
2. Initially the mobile agent is given the addresses of some other peers participating in the network. Typically the number of addresses provided should be small, and should be less than the branching factor. For best results these addresses should come from different sources. This increases the chances that these peers are operating in separate sub networks that are as yet unaware of each other.
3. The mobile agent clones itself enough times to allow a mobile agent to be sent to each of these peers.
4. After arriving at each peer the mobile agent decrements its journey-time, and updates the peer with information about its creator peer, and other peers encountered along the route so far. The mobile agent also updates itself with information on the current peer. If two mobile agents from the same creator peer arrive at the current peer within a preset time period, then the second mobile agent destroys itself. This may lead to less information being acquired about the network. However, it keeps the information up to date, and yet prevents peers that form cycles from having to deal repeatedly with mobile agents from the same creator peer.
5. If the mobile agent's journey-time has expired, then it returns to its creator peer, and updates its creator peer with all the information it has collected on its journeys. Otherwise the mobile agent clones itself enough times to allow a clone of itself to be sent to each peer that was known to the current peer before the mobile agent arrived. By excluding previously known peers, the incidence of revisits is reduced.
6. The process is continued from step 4.

4.4. Implementation and Architecture:

We implemented our mobile agent based solution using the Aglet Software Development Kit (ASDK) [37] with the patch supplied by K. Muniandy [38] to make it compliant with our Java 1.3. The general architecture of our proposed solution is built upon the existing Aglets architecture. The solution developed by us is an Aglet based protocol to be used as part of a larger solution, rather than a stand-alone application. Figure 1 below shows the class diagram of the general architecture of our solution:

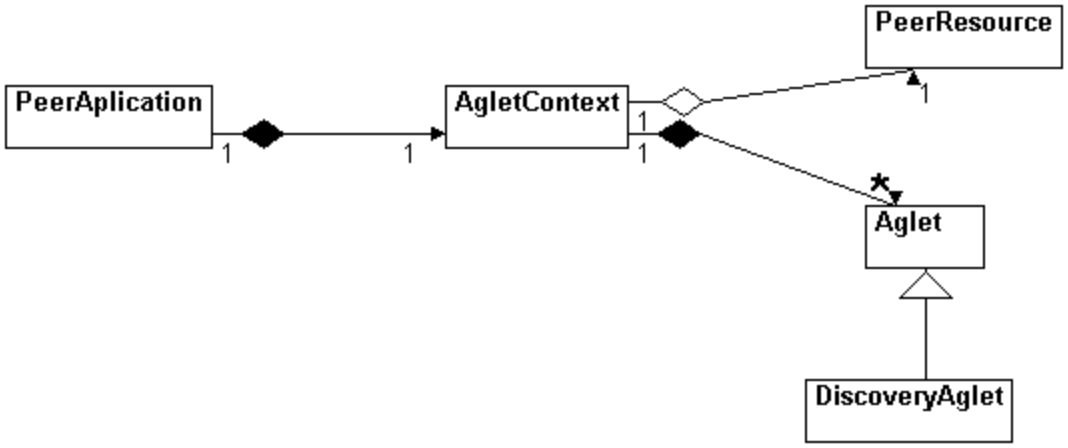

Figure 1. Class Diagram

DiscoveryAglet is the mobile agent that travels the network, and extends the Aglet class provided by the ASDK. It implements the peer discovery algorithm defined above, and the methods that do this cannot be overridden. However, the DiscoveryAglet may be extended to allow it to perform some specific actions related to the types of peers that it is searching for. The ASDK implements the mobility of its mobile agents by serializing them. So the Aglet takes its state with it by using its instance variables. Therefore to keep our DiscoveryAglet as small as possible all the instance variables are of type boolean, int, String, or Vector. As these classes are all part of the standard Java Virtual Machine it means that their bytecode does not have to be transmitted with the DiscoveryAglet, thus saving time and bandwidth.

The **AgletContext** class is part of the ASDK, and is needed to maintain and manage Aglets within a secure environment. Aglets can only communicate with other objects that are within the AgletContext. Therefore, the AgletContext provides methods allowing other objects to be placed within it, and retrieved from it. We did not need to modify this class.

PeerResource is an object that holds the information about all resources at the current peer. After it has been initialised, it is placed within the AgletContext. Therefore, the mobile agents within the AgletContext can access it. The PeerResource class contains certain methods that cannot be overridden to provide the mandatory functionality of a resource as described above. Further functionality was added based on some features of the Simple Network Management Protocol (SNMP) [39]. In particular the resource object can store objects using a String as the key to them, retrieve an object when given the correct key, and provide an enumeration of keys and values. The PeerResource may be extended to allow particular types of peers to add specific functionality for particular types of mobile agents.

The **PeerApplication** is a stand-alone java application that encapsulates all the software components referred to above. It is possible to use Tahiti, the default mobile agent server that comes with the ASDK, as well as a stationary agent that can be used to manipulate the

PeerResource. However we created our own EmbeddedAgletServer class containing an embedded mobile agent server, and all the functionality needed to use the DiscoveryAglets. This allows an application builder to quickly and easily incorporate our mobile agent based solution into their P2P applications.

4.5. Security:

Security is generally a problem with mobile agents. In our work we did not implement any specific security precautions per se, because we felt that we did not need security beyond what is provided by the Java Virtual Machine and the ASDK security model. We did however use many of their features in different ways.

Due to the nature of the solution security restrictions cannot be specified in terms of the code base because it cannot be known in advance where the DiscoveryAglet will come from.

When an Aglet, that is not an instance of the DiscoveryAglet, arrives at the peer, it can easily be destroyed immediately. Also the DiscoveryAglet does not need any security privileges once it is received. The only resource that it needs to access is within the AgletContext.

There may be specific application areas where the PeerResource contains complex objects that need access to local resources. This can be achieved by the use of privileged code within these complex objects.

So overall we deem that the security threat posed by the DiscoveryAglets is similar to that posed by Applets.

5. Conclusion:

In this paper we looked at the general problem of resource discovery on P2P networks. We looked at some of the current solutions, and examined some of their strengths and weaknesses. We then proposed a mobile agent based solution. We discussed the advantages of this solution. We described its architecture and its implementation using Aglets.

While there are several areas of the work presented here that require further investigation, there are two that particularly interest us. Firstly, we would like to assess the performance of our proposed solution in a large uncontrolled network, because so far all our testing has been in a controlled laboratory. Secondly, we would like to develop more mobile agents that are more application specific and which deal with increasingly complex resources that are defined using XML.

Finally from our work to date we believe that mobile agents do provide a viable means of performing network resource discovery.

6. Acknowledgements:

I would like to thank Dr. David Gray for introducing the concept of mobile agents to me, and for his help and advice in the preparation of this paper. Thanks also to my family for their continual encouragement and support.

7. References:

[1] Shirky, C, *What Is P2P... And What Isn't*, O'Reilly Network, 24/11/2000
[2] Napster, Inc., http://www.napster.com/
[3] SETI@Home, http://setiathome.ssl.berkeley.edu/
[4] ICQ, Inc, http://www.icq.com/
[5] eMikolo Networks, Inc., http://www.emikolo.com/
[6] Groove, Groove Networks Inc., http://www.groove.net/
[7] Caglayan, A. and Harrison, C. *Agent Sourcebook*. Wiley Computer Publishing, 1997.

[8] Green, S. et al., *Software Agents: A review*, Department of Computer Science, Trinity College Dublin, 1997
[9] Danny B. Lange and Mitsuru Oshima, *Seven Good Reasons for Mobile Agents*, Communications of the ACM, Vol. 42, No.3, March 1999.
[10] Dan Connolly, Mobile Code Systems, http://www.w3.org/MobileCode/
[11] General Magic, Inc., http://www.genmagic.com/
[12] IBM Japan Aglets, http://www.trl.ibm.com/aglets/index.html
[13] Java, Sun Microsystems, Inc., http://java.sun.com/
[14] Aglets Community, http://aglets.sourceforge.net/
[15] Yahoo! Inc., http://www.yahoo.com/
[16] Google Inc., http://www.google.com/
[17] Charney, B, *The World Wide $#@%@$ing Web!*, ZDNet News, 23/12/2000
[18] OpenNap: Open Source Napster Server, http://opennap.sourceforge.net/
[19] Scholl, Napster protocol specification by analysis, http://opennap.sourceforge.net/napster.txt
[20] Findlaw, http://www.findlaw.com/napster/index.html
[21] Truelove, K, *OpenNap Use Crashes*, O'Reilly Network, 11/05/2001
[22] Gnutella, http://www.gnutella.co.uk/
[23] BearShare, http://www.bearshare.com/
[24] Limewire, http://www.limewire.com/
[25] Clip2, *The Gnutella Protocol Specification v0.4, Document Revision 1.2*
[26] Adar, E, and Huberman, B, *Free Riding on Gnutella*, First Monday, Vol.5, No.10, October 2000
[27] Clip2, Reflector White Paper, http://dss.clip2.com/reflector_wp.html
[28] KaZaA, http://www.kazaa.com/
[29] FastTrack, http://www.fasttrack.nu/
[30] Sun Microsystems, Inc., http://www.sun.com/
[31] Project JXTA, http://www.jxta.org/
[32] Gong, Li, *Project JXTA: A Technology Overview*, Sun Microsystems, Inc., 25/4/2001
[33] Sun Microsystems, Inc., *JXTA v1.0 Protocols Specification, Revision 1.1.1*, 12/6/2001
[34] *Extensible Markup Language (XML) 1.0*, Second Edition, http://www.w3.org/TR/2000/REC-xml-20001006
[35] Sedgewick, Robert, *Algorithms in C++*, Addison-Wesley,1992
[36] J. Kleinberg, S.R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, *The web as a graph: Measurements, models and methods*, Proceedings of the International Conference on Combinatorics and Computing 1999.
[37] Aglets Software Development Kit, 1.2.0, http://sourceforge.net/project/showfiles.php?group_id=7905
[38] Muniandy, K, Modified aglets1_2.jar, http://www.cis.ohio-state.edu/~muniandy/agletsd-j2/1_2/
[39] Marshall T. Rose, *The Simple Book*, Prentice-Hall International, 1991