Large Scale Dissemination Using a Peer-to-Peer Network

Konstantinos G. Zerfiridis
Department of Informatics
Aristotle University of Thessaloniki, Greece
zerf@csd.auth.gr

Helen D. Karatza
Department of Informatics
Aristotle University of Thessaloniki, Greece
karatza@csd.auth.gr

Abstract

The widespread use of broadband networks and the evolution of Peer-to-Peer systems changed dramatically the way Internet is used today. P2P file sharing networks are one of the most popular ways of sharing and distributing new content. But along with the benefits of these networks, certain patterns became apparent. It could take a long period of time for new, highly anticipated files to become available for download, due to high demand. Therefore, the use of such networks as a mean of file dissemination is not always successful, especially when the files are of considerable size. In this paper Peercast is presented, a P2P dissemination system, along with simulation results. Our focus in this paper is on how this and other P2P file sharing networks can be configured to optimize the dissemination process of highly anticipated files.

1. Introduction

As the average bandwidth capacity increases, users around the world demand shorter response time. While the servers are able to acquire more bandwidth, they can not keep up with the rapidly increasing requests of the users. When any file of considerable size has to be disseminated to a numerous amount of receivers, the network can be saturated quickly, clogging the host computer. Such is the case, for example, when any highly anticipated software is released and several people are trying to download it at the same time. This became known as the middle night madness problem [1] because that is the time new software are made available, in order to avoid congestion.

As today's needs for data transfer steadily increase, traditional ways of making data available to the masses become obsolete. Conventional FTP servers can no longer serve as a way of distributing large amounts of data. For example, modern Linux distributions can span more than one CD. Assuming that the server's bandwidth is 1 MBit/sec and the requested software is distributed in 2 ISO CD images, the server could only serve about 50 clients in a period of one week even in the theoretical case that no errors occur. Mirroring the required content on several dispersed servers, cannot always compensate for the rapid traffic increase.

The main architecture used for casting data through the Internet is IP multicast, which mainly targets real-time non-reliable applications. It extends the IP architecture so that packets travel only once on the same parts of a network to reach multiple receivers. A transmitted packet is replicated only if it needs to, on network routers along the way to the receivers.

Although it has been considered as the foundation for Internet distribution and it is available in most routers and on most operating systems, IP multicast has not so far lived up to early expectations. Its fundamental problem is that it requires that all recipients receive the content at the same time. The most popular solution to this problem was to multicast the content multiple times until all of the recipients obtain it. Some of the other drawbacks of IP multicast include small address space (26-bit), need of large routing tables and lack of congestion control and reliable transfer control.

Several algorithms arise for membership management and packet replication to solve problems such as server implosion from client side NACKs (negative acknowledgments), server explosion from maintaining status of the download process for each client and managing downloads requests by users connected with different bandwidths. Forward Error Correction (FEC) has long been used for the dissemination of static data as it provides graceful degradation of performance in the presence of packet losses. Its greatest disadvantage is that it is very demanding on CPU and memory [2].

Although IP multicast might be considered ideal for applications that require relatively high and constant throughput but not much delay, it is not suitable for applications that may tolerate significant delays but no losses. This is the case with file distribution. These days, a new way of disseminating files emerged. File sharing networks [3] are perhaps the most commonly used

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 7, 2009 at 06:18 from IEEE Xplore. Restrictions apply.

Peer-To-Peer applications. P2P systems existed since the birth of the Internet, but as bandwidth, computational power and great storage capacity came to the masses, their popularity increased. Such systems have been used for diverse applications: combining the computational power of thousands of computers, forming collaborative communities, instant messaging, etc.

P2P file sharing networks' main purpose is to create a common pool of files where everybody can search and retrieve any shared files. But along with their popularity several problems emerged. A study conducted at the Xerox Palo Alto Research Center showed that 70% of Gnutella users provided no files or resources to the system and that 1% of the users were providing half of the total system resources [4]. This created network bottlenecks causing further inter-domain jamming.

File sharing networks had never been designed for file dissemination. Nevertheless, people turn to them to find highly anticipated software or even video files, when the official server stops responding due to high demand. Although extensive research has been done about how existing P2P networks operate over time and how they can be optimized [5, 6] the dissemination process of highly anticipated files over such networks remains unexplored. The purpose of this paper is to present Peercast, a network that is designed to assist in file dissemination and to show simulations and conclusions that could potentially benefit existing P2P file sharing networks.

The structure of this paper is as follows. Section 2 introduces PeerCaster, the agent based platform used. In section 3 the suggested approach is described. Section 4 shows the simulation model of the system. The results and drawn conclusions are summarized in section 5 and finally, section 6 presents suggestions for further research.

2. Background

Software agents are programs that act on behalf of clients. They are able to perform predefined tasks that are assigned to them. This is done either with or without the supervision of the user, depending on the given job.

Mobile agents have an additional property [7]. The ability to transport themselves on different systems after being executed, carrying with them their program code, current state of execution and any data which was obtained. This gives them the unique capacity of living on a distributed network rather than on a distant stationary system, and to take advantage of the services that each host has to offer locally. Furthermore, mobile agents allow proprietary code to be used on the hosts, allowing complete customization of the retrieved results. The unique properties of the mobile agents give them the edge in comparison to the traditional client-server paradigm. The hosts implement a specified environment that can authenticate the origin and credentials of the arriving mobile agents, provide for them the necessary execution machine and limit their access to system resources [8].

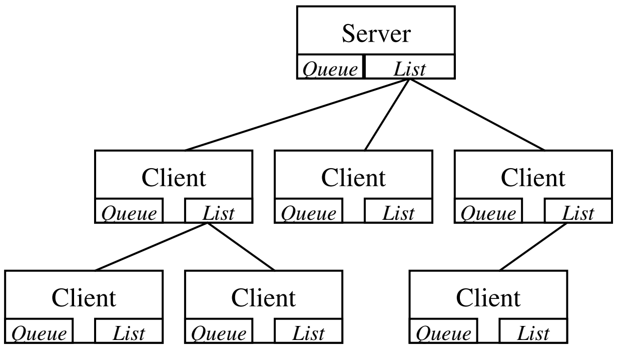

Figure 1. The dissemination network

Mobile agents have been used in the past instead of protocols [9], for file transfer [10] and as a dynamic system for information discovery and retrieval [11]. There are many applications that would benefit from the use of mobile agents as a vehicle for getting around bottlenecks. PeerCaster [12] is a platform implemented in Java that uses mobile agents as a vehicle delivering great amount of static data to users on a heterogeneous network. This is done by splitting the data into small packets, loading them onto mobile agents and releasing them to the peers where the payload is delivered and continue according to their itinerary. The coordination and communication overhead is acceptable considering the scalability that can be gained by the nature of the agent based system. As they can operate asynchronously and independently of the process that created them, they do not need to report back to the server.

In this paper, PeerCaster was used as a mean of distributing high-demand files without clogging the host computer. This system could be integrated as part of a P2P file transfer network, or it could be used as an alternative to multicast for large files with great demand, such as the release of a new version of popular software as depicted in [1].

3. The Network

When a file needs to be downloaded by more clients than the server can handle, alternative algorithms have to be utilized. The naive way of avoiding retransmissions is to pipeline the file through all the clients. But this is not a viable solution because clients might have to indefinitely wait to be served.

The proposed algorithm uses a dynamically changing tree of clients (figure 1) in order to avoid uneven flow of data and intersperse congestion points which can compromise inter-domain quality of service. The server

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 7, 2009 at 06:18 from IEEE Xplore. Restrictions apply.

can upload the file to a certain number of clients simultaneously. When the server successfully uploads a file to a client, it keeps a reference of this client to a short (up to 100 entries) FIFO list. As new clients are served, the list is enriched with newer clients and older clients are removed in order to avoid server explosion. Although the server has a small queue (of up to 10 clients), most of the clients are expected to find this queue filled. This is the case especially at the beginning of the dissemination process, as clients arrive more rapidly than the server can handle. When this happens, the server sends to the client the list of clients that already downloaded the file. This way, the new client can download the file from a client that was already served, removing the congestion from the server.

When a client finishes the download, it acts as a server for other clients. Similarly to the server, the clients have a short queue. If a client A requests the file from a client B that has it, and that client B can not serve client A immediately, A is queued. If the queue is full, client B sends its own list of served clients (up to 20 entries) to client A, so that it can continue searching. If a client is not able to be served or queued, it retries after a certain period of time to contact the server.

Several issues arise about the performance of this algorithm in a heterogeneous network. For example, what is the benefit of allowing several clients to download from a single peer? It will reduce the average waiting time, but what consequences will it have on the downloading speed and in the long run on the total number of served clients? How can the size of the queue in each client affect the dissemination process?

4. Simulation model

In this section details are presented about the simulation model for the proposed network, and it is shown how different strategies might affect the dissemination process. The system was populated with clients arriving according to the exponential distribution. The simulation period was set to be 2 weeks (1209600 seconds). During the first week the mean interarrival time was incremented linearly from 5 to 20 sec in order to simulate demand on a highly anticipated file. For the second week the exponential distribution was used with 20 sec mean interarrival time. The file size was set to be 650MB (the size of a full CD).

All the clients that populated the system were set to have broadband connections to the Internet, resembling cable modems and DSL. This is done in order to use a realistic model. As in many cases, such connections have different download and upload speeds, four different categories of users were used. The first category (10% of the clients) had download and upload speed of 256 Kbps, the second (40% of the clients) had 384 Kbps and 128 Kbps respectively, the third (20% of the clients) had 384 Kbps download and 384 Kbps upload speed, and the fourth (30% of the clients) had 1.5 Mbps and 384 Kbps respectively. This configuration is a theoretical model, and is used to compare how the same network performs under different conditions.

These kinds of clients are always on-line. However, they are not expected to share the file for ever. Therefore they were set to leave the dissemination network with exponential distribution and mean time of four days. Additionally, some clients are expected to refuse to share the file. Therefore, 10% of the clients were set to leave the dissemination network immediately after they download the file.

The server was set to be a DSL user as well; having 1.5 Mbps download / 384 Kbps upload connection (fourth category) to the net and never to going off-line. As the server is only uploading files, the simulation would have given the same results if the server had 384/384 connection to the net (third category). An additional difference between the server and the clients is that the server keeps a more extensive list (100 entries) of clients that it served, whereas the clients have a relatively shorter list (20 entries).

The actual connection speed between two clients is calculated at the beginning of each session, taking into consideration the theoretical maximum speed they could achieve and an exponentially distributed surcharge, in order to simulate additional network traffic and sparse bottlenecks. If a new client cannot be served or queued immediately, it waits for 600 seconds and retries.

Our focus is on how to use the server's and the clients' resources in an optimum way to serve as many clients as possible in a certain period of time. As it was mentioned earlier, the behavior of this network can change dramatically under certain conditions. The system’s performance is investigated at the beginning (2 weeks) of the dissemination, under different conditions.

When a client is serving only one peer at a time, the mean response time is expected to decrease. But by not allowing multiple uploads, the mean response time will increase as new clients enter the system. On the other hand, a client’s bandwidth is best utilized when serving multiple peers simultaneously. This will definitely increase the mean service time as the bandwidth of a serving client will have to be shared among several peers. But in the long run, more clients will be served within the same period of time. In order to determine how the number of simultaneous uploads can affect the system, four simulations were carried out where 1, 2, 4 and 8 clients respectively were served by each peer at a time.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 7, 2009 at 06:18 from IEEE Xplore. Restrictions apply.

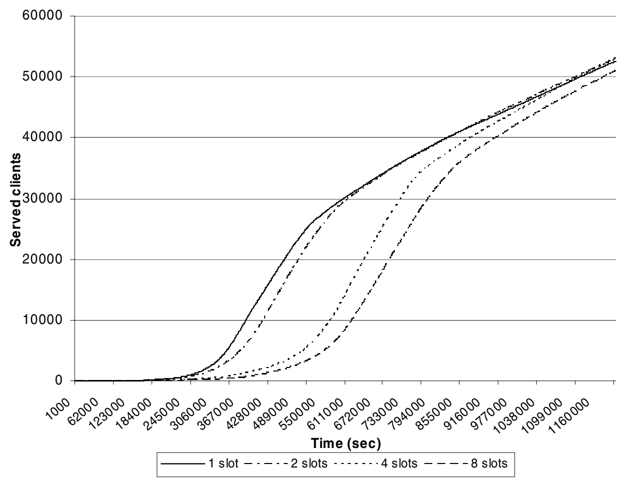

Figure 2. Number of served clients over time, under different number of simultaneous uploads

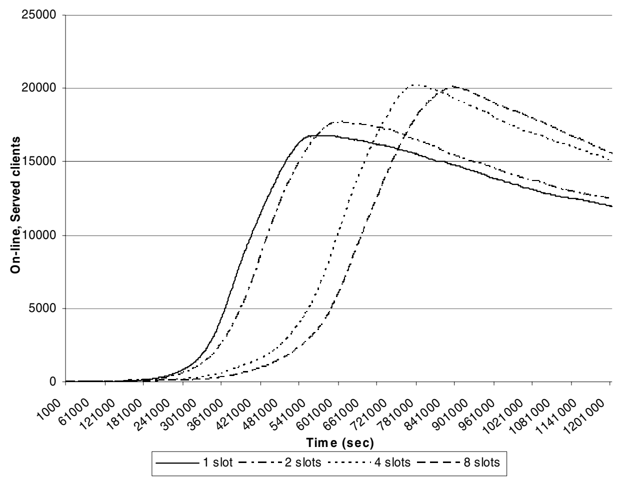

Figure 3. On-line clients that have been served, under different number of simultaneous uploads

Another parameter that could affect the dissemination process is the size of each client's queue. Shorter queues can increase the mean response time. That's because arriving clients that find an available peer to be queued on, will be served in a shorter amount of time if the queue is smaller. But in that case, at the beginning of the dissemination process most of the clients will not be able to be queued on a peer. This will cause them to enter in several timeout loops, leading to unfair treatment in some cases (a client that arrives later being served first). Again four tests were carried out, where the participating clients had queue sizes equal to 1, 2, 4 and 8.

5. Results and conclusion

As it is shown in figure 2, the number of simultaneous uploads affects significantly the population of the served clients. More specifically, at the beginning of the dissemination, using just one slot seems to speed up significantly the creation of a critical mass of served clients. The critical mass is the point where the rate of served clients in the system starts to decline. That happens when the rate of arriving and the rate of departing clients balance out.

This can be explained because when serving just one client at a time, all the bandwidth is dedicated to serve faster that peer. As in theory the population of served clients is expected to increase exponentially, it is of great significance to have several served clients in the system as soon as possible.

On the other hand, by using just one slot, a significant percentage of bandwidth might go to waste. As figure 2 shows, serving 2 clients at once will not greatly diminish the performance of the system. Nevertheless, it delays the system from reaching the critical mass of clients. And in the case that even slower clients (dial-up users) join in, the clients would have to use more slots to utilize their full bandwidth. Figure 3 shows that using more uploading streams, increases the number of serving-clients needed to reach the critical mass. That's because using multiple streams increase the mean response time at the beginning of the dissemination. However, as more clients are served simultaneously, the number of clients finishing the download increases rapidly.

Additionally, figure 4 reveals that the size of the queue plays a significant role, especially after the critical mass is reached. At the beginning of the dissemination process, the system behaves better as the queue size increases. But when the number of clients reaches a certain point, the system's performance seems to decrease. On the other hand, a smaller queue gives much better results in the long run. This behavior occurs because when an adequate number of served clients exist on the system, it is more favorable for an arriving client to keep searching for a free peer than to be queued on a long queue early on. As shown in figure 5, the size of the queue does not affect significantly the time period in which the balance occurs. That's because the bandwidth utilization cannot be affected directly by the queue size.

In order to compensate for these shortcomings, the Peercast system utilizes dynamically changing number of slots and queue size for the clients. As a mean of keeping track of the number of served agents, the server instructs a small percentage (5%) of the arriving clients, to send back a message when they finish downloading the file. This way the server can estimate when the critical mass will be reached. This information is then passed on to each arriving client so that they know, without contacting the server, when to increase the slot number and decrease the queue size.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 7, 2009 at 06:18 from IEEE Xplore. Restrictions apply.

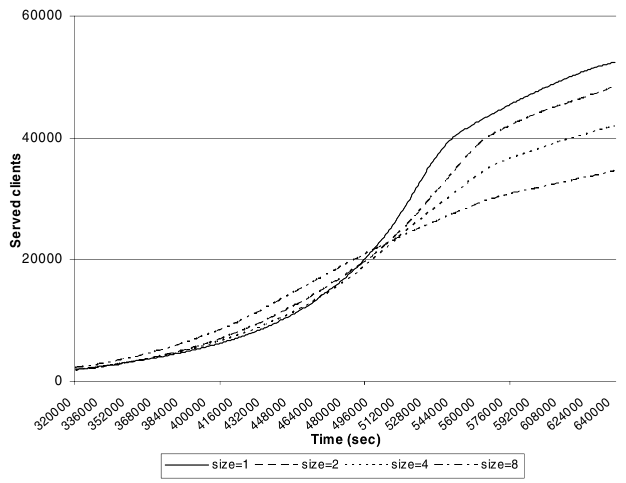

Figure 4. Number of served clients over time, under different queue sizes

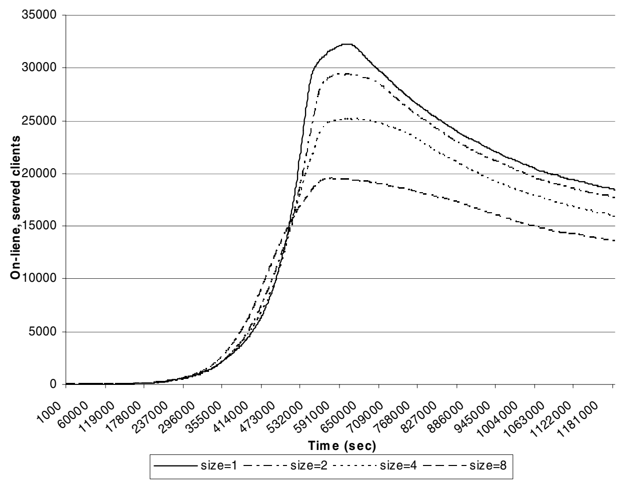

Figure 5. On-line clients that have been served, under different queue sizes

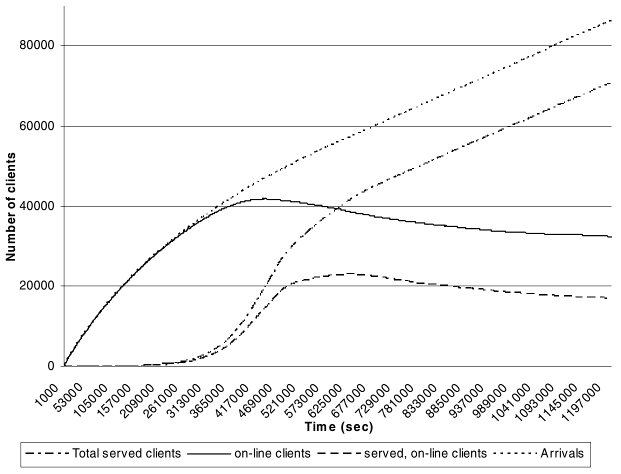

Figure 6. Status of the client population over time in the Peercast system

Additionally, in Peercast a client uses as many slots as needed in order to utilize its full bandwidth, but it favors the fastest peer by giving it all the bandwidth that it can handle. This way clients use their full bandwidth and new serving clients are created with faster rate. Although this policy is proved to be unfair for slower clients, it can be used as a way to build quickly a vast backbone tree of clients at the beginning of the dissemination. When the critical point is reached, the queue size drops from 8 to 2 entries, the slots become 4 and the peers are treated equally. Using more slots significantly increases the mean response time, as seen in table 1. A simulated representation of the population of clients at any given time in the first two weeks is depicted in figure 6.

Table 1 and table 2 show that the mean response time is increased in all cases. That's mainly because the clients that arrive early on the dissemination process have to wait for a long period of time to be served. When the rate of arrivals balances with the rate of clients being served, the mean response time stabilizes to lower levels. Therefore, clients arriving later in the system benefit from a faster service. This is depicted in figure 7.

Further simulation results, not shown here due to space limitation, reveal that the size of the list of served peers that each client has, does not significantly affect the dissemination process. The same conclusion was reached about the timeout period each client has to wait before retrying to find a client to be queued on. However, the time period in which the critical mass is reached is highly depended on the mean interarrival time and the heterogeneity of the clients. Therefore it can only be estimated after an appropriate period of time.

Existing P2P file sharing networks cannot obviously be rebuilt to optimize the dissemination of new files.

Table 1. Mean response time (queue=10)

	256/256	*384/128*	*384/384*	*1.5/384*
1 slot	221979	218498	216814	216232
2 slots	261779	249400	250301	227057
4 slots	387713	362996	363264	306762
8 slots	449403	415403	414127	334793

Table 2. Mean response time (slots=2)

	256/256	*384/128*	*384/384*	*1.5/384*
queue=1	215911	203957	203612	181570
queue=2	226016	214102	213228	192832
queue=4	239629	229174	230197	207302
queue=8	255650	242378	239816	216147

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 7, 2009 at 06:18 from IEEE Xplore. Restrictions apply.

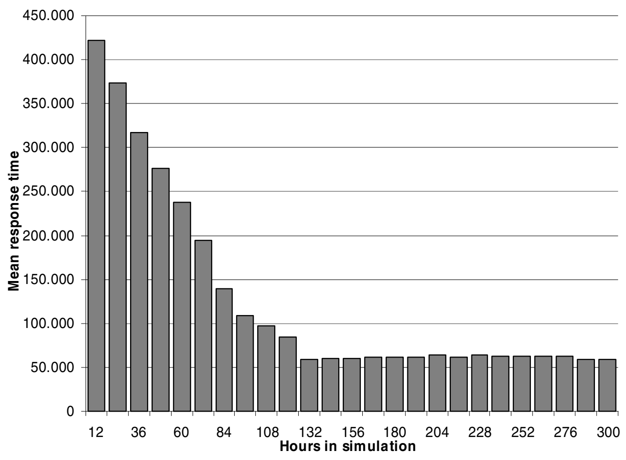

Figure 7. Mean response time of clients that arrived in each time period (12 hours)

However, some changes to the application level are expected to increase their performance. For example, in the Gnutella network, each application could perform a search to determine which of the files that it currently shares are unique or they are only found on limited number of peers. By monitoring the search queries it can resolve which of those files are most popular. In case a file is found to be a "hot-spot", priority over the bandwidth should be given to any peer that requests it. If multiple peers request it, it should be given to the one that shares the largest number of files as this peer would most likely share the file with other peers. Simulation results of this scenario are pending.

6. Future Work

For the current P2P network implementation we used a monolithic approach: all the data has to be sent to a client, before this client starts sending it to another peer. The PeerCaster platform is highly scalable because it was implemented using mobile agents. A new version that replicates groups of 256KB packets, to adjacent peers as they arrive, is under way. This is expected to alleviate the problems that are caused from peers that go off-line immediately or soon after they finish downloading the requested file. The synchronization between the peers is done in predetermined time intervals, called epochs [13]. The peers are segmented in virtual groups according to their bandwidth and the epoch size depends on an estimation of the minimum bandwidth between the peers that form each dissemination group. Simulation results from this network are expected to show alleviation of several issues raised in this paper such as the increased mean response time at the beginning of the dissemination. Additionally, distributions varying with time were incorporated for more realistic long-run simulations, as depicted in [14]. We are also working towards creating a version that uses prior knowledge of a peer's content to push newly arrived packets and utilize software FEC.

7. References

[1] E. Schooler, and J. Gemmell, "Using Multicast FEC to solve the Midnight Madness Problem", *Technical Report*, Microsoft research, September 1997.

[2] L. Rizzo. "On the feasibility of software FEC", *Technical report*, Univ. di Pisa, Italy, January 1997.

[3] M. Parameswaran, A. Susarla, and A.B. Whinston, "P2P Networking: An Information Sharing Alternative", *Computer Journal*, IEEE Computer Society, Vol. 34, July 2001, pp. 31-38.

[4] E. Adar, and B.A. Huberman, "Free Riding on Gnutella", *Technical report*, Xerox Palo Alto Research Center, 10 August 2000.

[5] E.P. Markatos, "Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella", *In the Proceedings of the CCGrid 2002,* Second IEEE/ACM International Symposium on Cluster Computing and the Grid, May 2002, pp. 65-74.

[6] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella Network: Properties of large-scale peer-to-peer systems and implications for system design", *Internet Computing Journal*, IEEE Computer Society, January 2002, pp. 50-57

[7] D.M. Chess, B. Grosof, C.G. Harrison, D. Levine, C. Parris, and G. Tsudik, "Itinerant Agents for Mobile Computing", *Journal of Personal Communications*, IEEE Computer Society, Vol. 2, No. 5, October 1995, pp. 34-49.

[8] D.M. Chess, C.G. Harrison, and A. Kershenbaum, "Mobile Agents: Are They a Good Idea?", *Research report*, IBM T.J. Watson Research Center, Yorktown Heights, New York, March 1995.

[9] B. Joy, "Shift from Protocols to Agents", *Internet Computing*, IEEE Computer Society, Vol. 4, No. 1, January 2000, pp.63-64.

[10] T. Spalink, J.H. Hartman, and G. Gibson, "The Effects of a Mobile Agent on File Service", *In the Proceedings of the First International Symposium on Agent Systems and Applications*, Third International Symposium on Mobile Agents (ASA/MA '99), Palm Springs, California, IEEE Computer Society, October 1999, pp. 42-49.

[11] K.G. Zerfiridis, and H.D. Karatza, "Brute Force Web Search for Wireless Devices Using Mobile Agents", to appear in the *Journal of Systems and Software*, January 2004, Elsevier.

[12] K.G. Zerfiridis, and H.D. Karatza, "Mobile Agents as a Middleware for Data Dissemination", *Neural, Parallel &*

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 7, 2009 at 06:18 from IEEE Xplore. Restrictions apply.

Scientific Computations, Dynamic Publishers, Atlanta, Vol. 10, 2002, pp. 313-323.

[13] H.D. Karatza, and R.C. Hilzer, "Epoch Load Sharing in a Network of Workstations", *Proceedings of the 34th Annual Simulation Symposium*, IEEE Computer Society Press, SCS, Seattle, Washington, April 22-26, 2001, pp. 36-42.

[14] H.D. Karatza, “Task Scheduling Performance in Distributed Systems with Time Varying Workload”, *Neural, Parallel & Scientific Computations*, Dynamic Publishers, Atlanta, Vol. 10, 2002, pp. 325-338.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 7, 2009 at 06:18 from IEEE Xplore. Restrictions apply.