Optimized Dissemination of Highly Anticipated Content over an Itinerary Based P2P Network

Konstantinos G. Zerfiridis
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
zerf@csd.auth.gr

Helen D. Karatza
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
karatza@csd.auth.gr

Abstract

As the Internet evolved, peer-to-peer networks became one of the major enabling technologies. Alternative solutions for several issues emerged with the use o this technology. Such is the case for the dissemination of large files to several receivers. The traditional client-server approach can not keep up with the rapidly increasing requests of the users. P2P file sharing networks came as an unconventional way of relieving the server from becoming the congestion point. The purpose of this paper is to show and analyze simulation results of an itinerary based algorithm for the dissemination of a highly anticipated file. Emphasis is given on several parameters of the network and how they can affect its performance.

1. Introduction

New applications that use the Internet in innovating ways are appearing rapidly as broadband connections are becoming widespread. More users are now able to be constantly online through the use of inexpensive cable and Digital Subscriber Line (DSL) connections. Along with the benefits of having a faster connection to the net, certain patterns started to emerge. Traditional hypertext documents are giving way to rich media such as audio and video. The traditional client-server paradigm does not always come through when great amounts of data need to be disseminated to numerous broadband-enabled computers. Servers that publish highly anticipated files can become congested in a short period of time [1]. Mirroring the content has long being used as a way to alleviate the server from reaching its saturation point. This approach however is not always able to cope with the rapid traffic increase. This is especially the case when large files need to be disseminated to numerous users. Connection failures and retransmissions can add to the creation of network bottlenecks.

Perhaps the most commonly used P2P applications are the file sharing networks [2]. Their popularity can be attributed to their ease of use and high scalability. These networks started out as a way to create a big pool of files where anybody that participated had access to. However, early P2P file sharing networks became victims of their own success. Such networks were never designed for large file dissemination. Along with the increasing demand for rich media, another problem 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 [3]. Nevertheless people turned to P2P file sharing networks to find highly anticipated files, when the official server stopped responding due to high demand. This created network bottlenecks causing further inter domain jamming.

File sharing networks create in essence a common pool of publicly accessible files. Each user defines which files are to be shared. Centralized or decentralized search mechanisms allow for the participating users to search and retrieve a specific file. Files that are downloaded on a user's computer are considered as shared files. Depending on the algorithm used, these sharing networks can be divided into two groups. The first group consists of networks in which a file has to be downloaded completely before a user is able to share it to other users. The dissemination process of highly anticipated files on such P2P networks has been studied earlier [4]. The second group is comprised of a new breed of P2P applications. These applications became popular as they follow a different approach. In these networks, files are segmented into several smaller packets, allowing them to be distributed independently. This approach exploits a simple, yet powerful principle: All the clients have to contribute to the dissemination of the designated file, alleviating this way problems raised in [3]. As soon as a client starts downloading these packets, it starts offering them to the network at large. That means that a large file does not have to be downloaded in its entirety before it can be offered to other clients.

Extensive studies have been done about how file sharing networks of the first group operate over time and how they can be optimized [5, 6]. However, the dissemination patterns and the way that certain network parameters can affect the efficiency of the latter group remain unexplored. In this paper an itinerary based P2P dissemination network is introduced and a number of simulation results of different dissemination scenarios are presented in order to depict the network's behavior under a variety of conditions. Emphasis is given to the proposed itinerary based approach for large file dissemination.

The structure of this paper is as follows. Section 2 presents the structure of the proposed approach and shows the simulation model that was used. In Section 3 the simulation results are presented and analyzed, and finally in section 4 the drawn conclusions are summarized, and future research plans are presented.

2. The itinerary based 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. In this paper an itinerary based algorithm is used for file dissemination. This section depicts the details of this approach and presents the simulation model used.

As the Internet evolved, it became a place where anybody could easily make available to the public large files such as demo versions of applications and games or videos. Such releases are often followed by instant high demand making most economical broadband connections insufficient. Several P2P networks exist in which any given file has to be downloaded completely by a client before it can share the file to other peers. However, even these networks are often proved inadequate to cope with the high demand of popular large files at the beginning of the dissemination process. That's because it can take a long time before a sufficient number of users download the complete file and stay online to assist the rest of the users.

Another approach to this problem is to have the file segmented to several smaller parts and have them replicated among the clients that requested the file. This way, the clients can assist each other almost immediately. As an additional benefit, all the users contribute to the dissemination of the file, even if they go offline immediately after they finish the download. This approach is used by more recent P2P networks such as eDonkey [7] and BitTorrent [8].

In general, in such networks the server of a file, often called seed, in addition to sending packets to the clients, it maintains a list of the addresses of the clients that requested the file. When a new client arrives in the system, the server sends to the client this list. The client then takes over and sends requests to peers in this list to find those that are online and which packets each one has. It then requests the packets from the clients and it only refers to the server if it cannot find a packet anywhere else.

This approach is relatively straightforward, but it has several shortcomings. The clients always have an outdated list of peers. That is because the clients do not have knowledge of the existence of a peer that contacted the server for the first time after they did. Pulling the server frequently in order to have an updated list is not a viable solution as it would put additional load to the server. Therefore, when a client is unable to find a packet from any of the peers, it requests it from the server. This can overwhelm the server with too many requests that could have been avoided. Furthermore, a client arriving in the network has to request a list from a server. This adds to the server's heavy load and makes it a congestion point. After this, the client has to iterate through the peers to find one by one the appropriate packets in order to replicate the file.

These shortcomings have one common root. The algorithm described above relies on flooding queries to all peers and therefore it is in essence pull based. That is because each client has to find by itself the necessary packets that are scattered among the peers. Although flooding is simple and robust, it is not scalable. Additionally, no knowledge of which packets a peer already has is used in order to offer this peer a packet that it might not have. Furthermore, the server has no way of controlling how a packet will be distributed among the peers.

In order to address several of the above mentioned shortcomings, an itinerary based algorithm is proposed. Mobile agents [9] are used as carriers of the packets and additional information such as an itinerary [10] as depicted in figure 1. Furthermore, each client maintains a short list of peers, called database, in which the peers' network addresses along with the last known set of packets that they have are stored. This simple, yet powerful approach is expected to overcome several scalability issues.

Departing mobile agents from the server disseminate each packet to several destinations one-by-one. Additionally, they interact with each client's database in order to find which clients might need the packet that they are carrying. If anyone is found, the mobile agent checks if that peer is already part of its itinerary. If it is not, the agent includes that peer as a destination. However, the agents were set to have a maximum of 32 destinations.

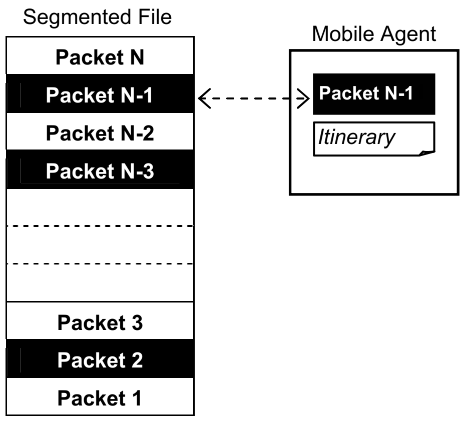

Figure 1. File segmentation

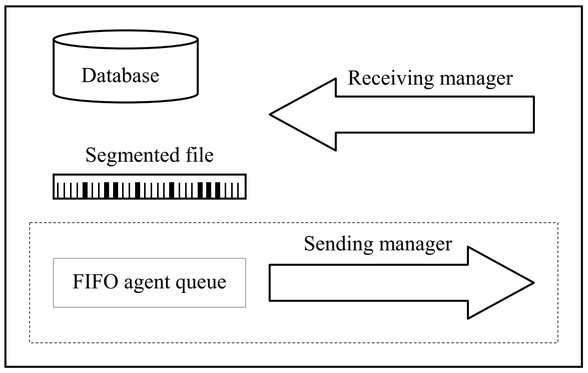

Figure 2. Client structure

The reason for that is because the itinerary has the potential to grow indefinitely. The obvious drawback for this is that the size of the itinerary can surpass the size of the packet. Furthermore, when the list of destinations grows beyond a certain point, the agent becomes useless. That is because by the time that the agent reaches most of the clients, they could have acquired that packet from another source. In this simulation, the packet size was set to be 1 Mbyte. In order to simulate dropped agents because of network failures, the agents were set to be able to visit 16 clients in average, using an upper bounded exponential distribution.

Each client, as depicted in figure 2, is comprised of three main modules: a peer database, a receiving manager and a sending manager. The peer database is responsible for maintaining content information about a limited number of peers. This database is periodically updated by pulling from the peers their latest list of received packets. In this simulation, the database was set to have a fixed size of 10 entries, and the time between updates was set to be 600 seconds. If during an update a peer is unreachable, its entry is removed from the database. A peer could also be removed from the database during the download process. If for example the client queries a peer in the database and the peer is unreachable, it will be removed. If there is an empty entry in the database, the client tries to find a new peer to be added. This is done by probing for peers the itinerary of any incoming agent.

An issue that arises is when a peer should be removed from the database. If a client's database was set to have the same peers throughout the entire time, the client would reach a point when it will not be able to find new packets from these peers. This issue was addressed by periodically estimating the amount of time it could take for the client to download all the available packets from the peers that are currently on its database. After this time elapses, the client iterates through the database to find the least useful peer by comparing its own contents to each peer's contents. The peer with the least amount of packets that this client needs is chosen for removal. A new peer is added when the next agent arrives, and the client estimates when this refinement should take place again.

This is not the case when the client has finished the download. At that point, the client acts as a seed for the given file. Therefore its main purpose should be to find peers in need of any packets. Since there are no incoming agents now, as the download has completed, new peers are added when a request for a packet is made to the client by an unknown peer. Again, the client estimates how long it would take to disseminate the packets to all the peers in the database that need them. After this time elapses, the client tries to find a new peer to add in the database and removes another if the database has reached its maximum capacity.

The receiving manager is mainly responsible for accepting or declining incoming agents. The manager can refuse the request of a remote agent to be transferred if the agent carries a packet that already exists on this client. In this case the agent moves on to its next destination. Another reason for declining a remote transfer request would be if the client doesn't currently have sufficient download bandwidth. If this happens, the agent continues with its itinerary and will retry to contact this peer later. If an agent cannot go to any of the peers that remain in its itinerary, it dies. An agent can also be lost if the connection between two peers is lost. If only a fraction of an agent is received, the client discards it. The receiving manager has also the responsibility to forward an incoming agent to the sending manager in order to continue according to its itinerary to the next client. Furthermore, the receiving manager monitors the client's download bandwidth. If it determines that it can handle the download of another agent simultaneously to the agents that it downloads currently, or if it doesn't download any agents at all, it refers to the database to find and request packets from other peers. If the receiving

Table 1. Clients' characteristics.

Groups	Participation	Download	Upload
1	10 %	256 Kbps	256 Kbps
2	40 %	384 Kbps	128 Kbps
3	20 %	384 Kbps	384 Kbps
4	30 %	1.5 Mbps	384 Kbps

manager cannot find any packets that it needs from the peers in the database, it requests one packet from the server. It should be noted here that when a client requests a packet from the server, the packet is randomly chosen from the list of packets that the client requires to complete the transfer. Simulation results have shown that if the packets were to be requested in any particular order, the performance of the network would decrease dramatically. That is because when all the clients request from the server packets in a specific order, they end up requesting the same packets. As a result, the clients in the network would have the same packets and therefore no packet replication among them would be possible.

The main responsibility of the sending manager is to utilize the upload bandwidth to the maximum. The sending manager maintains a FIFO queue of up to 10 outgoing agents. The manager is able to upload several agents to their destinations simultaneously in order to utilize all of the available bandwidth. It is also able to accept requests for a packet from another peer and create a mobile agent as a carrier of that packet. If the queue is full, remote requests for a packet are not accepted. However, agents forwarded by the receiving manager are always queued even if the queue has reached the given limit. Before an agent departs from the client, it contacts the database to find any additional peers that might need the packet that it carries. If any peers are found, they are added at the end of the agent's itinerary and the database is updated to indicate that those peers have the given packet. If the queue is empty, the sending manager iterates through the database to find peers that might need any packets. If two or more peers in the database need the same packet, one agent is created with multiple destinations.

The server acts as a client that finished the download. Clients refer to the server only on the beginning of the dissemination and when they are not able to find a packet from any of the peers in their database. Therefore it is essential that the server accommodates all the requests. However, it would not be possible to send all the packets that the clients in the network request, as this would overwhelm the server. On the other hand, in the proposed P2P network, one agent carries sufficient information for a new client to find other peers and begin the downloading process. Therefore, if a client requests a packet from the server, and the server's outgoing queue is full, the client's address is added to the itinerary of the

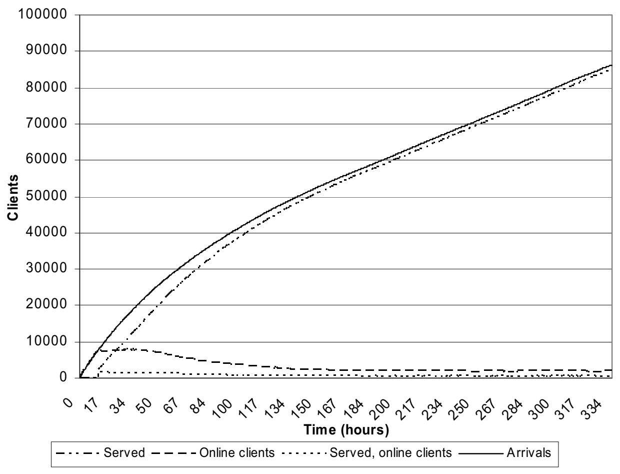

Figure 3. Network's state over time

first agent in the queue. This proved to have a dramatic increase in the performance of the network, alleviating along the way the server from maintaining a long queue.

The used simulation model was based on an object-oriented model of the network. 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 500MB.

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. Their participation in the population of the network and their upload and download bandwidth is shown in table 1. This configuration is a theoretical model, and is used to compare how the same demand is handled using different network parameters.

These kinds of clients are always online. However, they are not expected to share the file forever. Therefore they were set to leave the dissemination network with exponential distribution and mean time of 10 hours (36000 seconds). In order to simulate peers that are not willing to assist in the dissemination process, 10% of the clients were set to go offline immediately after they finish downloading the file. It is likely that this will significantly decrease the performance of the dissemination process. Nevertheless it is a behavior that can be expected. The server was set to have 1.5 Mbps download / 384 Kbps upload connection (resembling a DSL user) to the net and

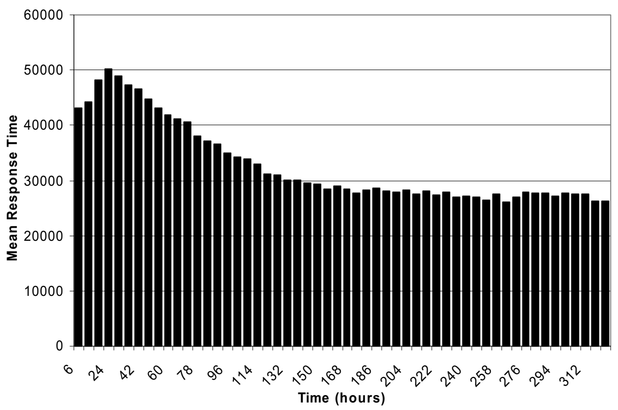

Figure 4. Mean response time in 12-hour intervals according to each client's arrival.

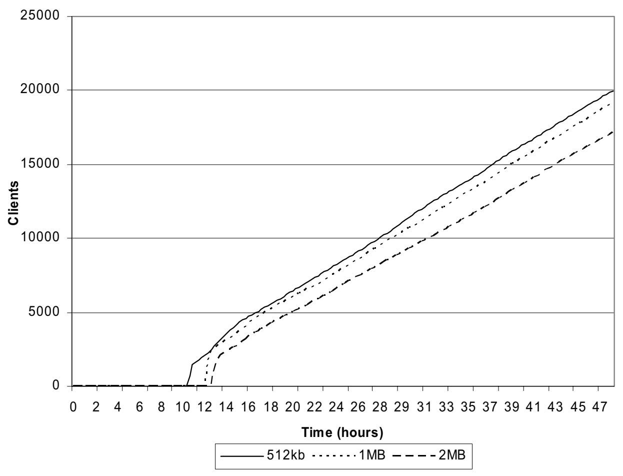

Figure 5. Served client over time, using different packet sizes.

never to go 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 group). 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.

3. Network parameters and simulation results

The network's performance is evaluated at the beginning (2 weeks) of the dissemination. As it can be seen in figure 3, the curve of served clients in the itinerary based approach follows closely the curve of the clients arriving in the system. This shows that the itinerary based approach can efficiently accommodate flash crowds. This can be explained as the server and clients treat each file segment individually. While the server is engaged by serving a peer, the rest of the peers replicate among each other the packets they already received. Having all the clients assist in the process from the early beginning proved to be a scalable system for large file dissemination. Coupled with the fact that the itinerary based approach uses prior knowledge of a peer's content to push packets, the initial waiting time is decreased significantly. Figure 4 shows the mean response time of the clients according to their arrival in the system, in 6-hour intervals. As it can be seen, clients arriving in the system after a certain period of time are accommodated faster. That is because there is a sufficient number of served clients that are online and serving other peers. On the other hand, early on the dissemination process there are not enough peers with packets to accommodate the arriving clients. However, as it can be seen in figure 4, clients that arrived at the beginning of the dissemination were served faster than the clients that arrived a little bit later. That can be explained as clients that arrived early may benefit by having higher probability of being in the server's database. Each agent departing from the server adds to its itinerary the clients in the server's database that need the packet that agent carries. Therefore, these clients are served faster, assisting along the way the clients in their own database. However, as more clients arrive in the system, the flow of agent is distributed throughout older and newer clients, increasing the mean response time.

Several issues arise about the performance of the algorithm under a variety of network parameters. For example, how can the size of the database affect the dissemination process? Does the client's sender FIFO queue size play a significant role in the overall quality of service? This section presents simulation results concerning how such parameters can affect the network's performance. In order to evaluate the efficiency of the algorithm in each case, a number of statistical measurements were calculated.

3.1 Packet size

The payload that each agent carries was set to be 1 MB. However, it would be interesting to see how the system behaves when smaller or larger packet size is used. Figure 5 shows that as the size of each packet get smaller, the amount of served clients over time increases. That's because more agents are in the system over time. Furthermore smaller agents can travel faster from one client to the other. Therefore packets and crucial information can be disseminated faster to new clients.

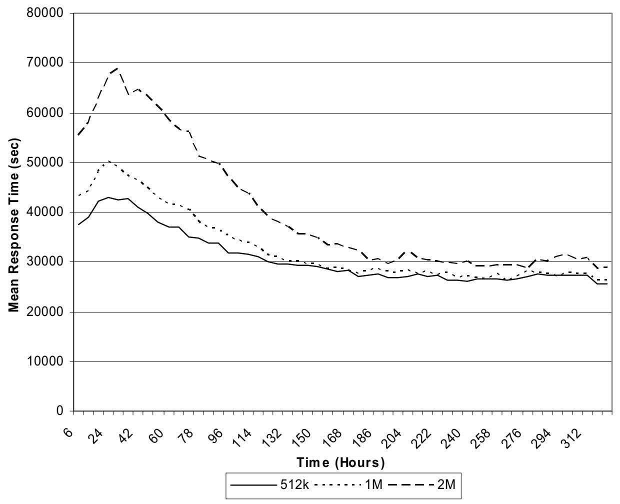

Figure 6. Mean response time in 6-hour intervals according to each client's arrival.

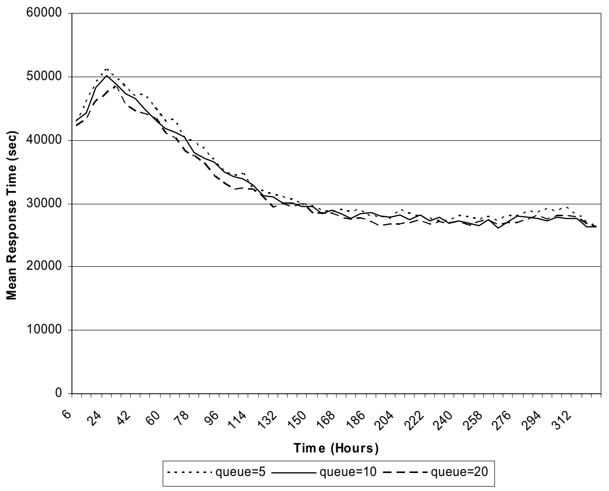

Figure 7. Mean response time in 6-hour intervals according to each client's arrival

Another observation is that the first served clients appear earlier when the packet size is smaller. That is because, as the size of the packet is getting bigger, the mean response time increases. This can be seen in figure 6 where the mean response time is shown, for the 3 simulation runs, in 6-hour intervals, according to each client's arrival. It should be noted that the 1MB and 512KB approaches seem to converge after a certain period of time. This means that when the system reaches a point, the size of the package can be increased without having a significant effect on the process. Another benefit of having agent carry smaller packets is that losing an agent has less impact on the client's performance. That's because, a lost agent is in essence wasted bandwidth and therefore, time. As the agents' payload is getting smaller, the amount of lost time because of lost agents decreases and therefore the performance of the network improves.

Table 2. The affect of the database size on the clients' performance.

Database Size	**5**	**10**	**20**
Mean Response Time	44911	35814	32760
Mean Time Waiting	18444	17325	16189

Table 3. The affect of the database's update time intervals on the clients' performance.

Database timeout period (sec)	**150**	**300**	**600**
Mean Response Time	35239	35511	35814
Mean Time Waiting	17029	17174	17325

3.2 Database size

As it was mentioned earlier, the itinerary based approach uses a database in order to maintain information about other peers. This database is frequently updated and new peers are added. It is interesting to see how the network's performance changes when the size of this database increases. The network was tested when a database of 5, 10 and 20 entries was used. Table 2 shows the mean response time and the mean time that a client has to wait until it is served. The mean response time is defined as the amount of time an arriving client spends online until it finishes the download. It becomes apparent that as the size of the database is getting larger the mean response time is getting smaller. That's because when a client has a large enough database, it can instruct each outgoing agent to visit more peers that are probably missing the packet that the agent has. This increases the overall performance of the network. A 20% decrease is observed in the mean response time when 10 entries are used instead of 5 for the database. This percentage drops to 8.5% when 20 entries are used instead of 10. This decrease is expected as the mean response time cannot decrease linearly. In the best case the mean response time can decrease asymptotically to the theoretical minimum time that the file transfer can take place. However it would not be sensible to have a very large database as the maximum size of an agent's itinerary is finite. Furthermore, the cost of maintenance of a large database could interfere with the client's primary goal of retrieving missing packets. The mean time that a client waits before the first agent arrives is also decreased. A drop of about 6% is observed in each case. This drop can also be attributed to the fact that as the database is getting larger, more clients are expected to be visited by agents that otherwise would not reach them.

3.3 Database update period

As it was mentioned earlier, the database in each client has a mechanism that periodically verifies the validity of its entries. A predetermined amount of time is set as the database's timeout period. When this time elapses the database checks to see if it contains a client that is no longer online. If that's true, this client is removed. Otherwise, a new list of packet that each client has is requested in order to update the database. It makes sense that the benefit of having an updated database is great. That's because if a client has old information about a peer, it might send a packet to that peer which was already acquired. Furthermore, a client that doesn't have current information about the contents of the peers in its database might not be able to find a packet that it needs. Table 3 shows the mean response time and the mean time that a client has to wait before the first agent arrives. Three simulation runs are shown for 150, 300 and 600 seconds timeout period for the database. As expected, it can be seen that both the mean response time and the mean waiting time is increasing as the timeout period increases. However, the increase in all cases is negligible (less than 1%). That's because the number of packets that the initial file was segmented and the size of the database, are sufficient to provide a client with enough resources throughout the dissemination.

3.4 Queue size

Each agent that arrives to a client has probably one or more destinations to its itinerary. The sending manager maintains a FIFO queue which acts as a buffer for the outgoing agents. In order to utilize the outgoing bandwidth to the maximum, this queue should be big enough to always have an agent waiting to be send. That way, when the sending manager estimates that there is enough bandwidth to initiate another agent transfer, it can find one in the sending queue. Additionally, as the sending queues are getting bigger, the mean response time should decrease. This is true as it can be seen in figure 7. That's because clients should be able to find service easier and therefore being served faster. However, as figure 8 shows, when using a smaller queue, the first clients in the dissemination process are served faster. That's because the agents are not forced to wait in long queues and are distributed faster. When more peers join the network, longer queues are needed to accommodate additional agents.

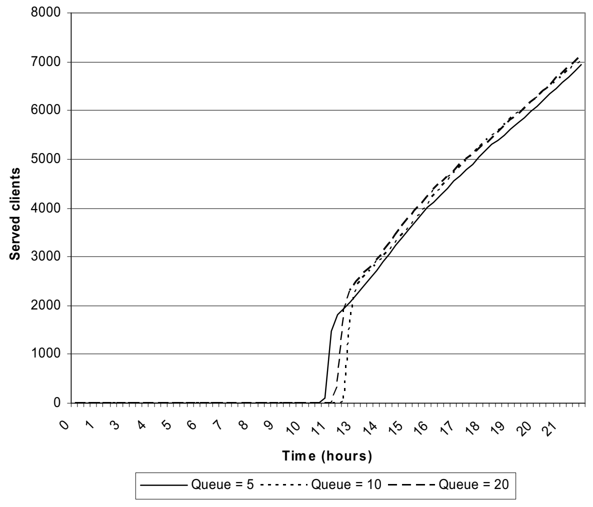

Figure 8. Served clients over time

4. Conclusions and future work

This paper addressed one fundamental challenge: the optimization of delivering a large file to several rapidly arriving clients. Coarse grain replication approaches for large files dissemination is often used to accommodate flash crowds efficiently. That is because the load can be redistributed among the participating peers right from the beginning of the dissemination. The presented itinerary based approach used mobile agents with a dynamic itinerary to deliver file segments to a number of clients. This approach has all the characteristics of pipelining along with the flexibility of a dynamic itinerary. Mobile agents have been used in the past instead of protocols [11] and for file transfer [12], but never as part of a P2P file sharing network. Their ability to transport themselves on different systems after being executed, carrying with them their program code and current state of execution gives them the unique capacity of living on a distributed network. As they can operate asynchronously and independently of the process that created them, they do not need to report back to the server.

The itinerary based approach presented in this paper could be integrated as part of a P2P file transfer network. It could also 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]. In such cases, where a so-called flash crowd overloads servers or networks and renders them useless, traditional ways of making data available to the masses do not apply.

As broadband-enabled computer users look to download larger files, a new breed of P2P networks gains popularity. An interesting simulation comparison would be between other packet replication based P2P networks and the itinerary based algorithm proposed here. Additional simulation experiments for the itinerary based algorithm presented in this paper are under way, using distributions varying with time for more realistic long run simulations, as depicted in [13]. Furthermore, an extension of the algorithm to incorporate synchronization between the peers in predetermined time intervals is under way. Two techniques are under consideration for the agent synchronization. The first is called location synchronization [14] and allows two or more agents to coordinate the location of their execution at various times. The second uses predetermined time intervals, called Epochs [15]. The peers are segmented in virtual groups according to their bandwidth and the synchronization time interval depends on an estimation of the minimum bandwidth between the peers that form each dissemination group. Neighbor selection policies are also examined. Simulation results from this network are expected to show decrease of the mean waiting time after critical mass is achieved.

5. References

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

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

[3] E. Adar and B. Huberman. “Free Riding on Gnutella”. Technical report. Xerox Palo Alto Research Center. October 2000.

[4] K.G. Zerfiridis and H.D. Karatza. “Large Scale Dissemination using a Peer-to-Peer Network”. In Proceedings of the 3rd International Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, IEEE/ACM International Symposium on Cluster Computing and the Grid 2003, Tokyo, Japan, 12-15 May 2003, pp. 421-427.

[5] E.P. Markatos. “Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella”. Proceedings of the CCGrid 2002, 2nd IEEE International Symposium on Cluster Computing and the Grid. May 2002. pp. 65-74.

[6] Ripeanu M., 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] The Donkey Network. What is eDonkey. December 2003. Available from: http://www.thedonkeynetwork.com

[8] BitTorrent. A tool for distributed download. December 2003. Available: http://bitconjurer.org/BitTorrent/

[9] 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.

[10] 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. October 1995. Vol. 2, No. 5, pp. 34-49.

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

[12] T. Spalink, J.H. Hartman and G. Gibson. “The Effects of a Mobile Agent on File Service”. 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.

[13] H.D. Karatza “Task Scheduling Performance in Distributed Systems with Time Varying Workload”, Neural, Parallel & Scientific Computations, Dynamic Publishers, Atlanta. September 2002, 10(3): 325-338.

[14] S. Mishra and P. Xie. “Interagent Communication and Synchronization Support in the DaAgent Mobile Agent-Based Computing System”. IEEE Transactions On Parallel And Distributed Systems. March 2003. Vol. 14, No. 3, pp. 290-306.

[15] 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.