Dissemination Scenarios in Peer-to-Peer Networks

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
zerf@csd.auth.gr

Abstract

As the average bandwidth capacity is increasing, users around the world demand for shorter service time. While the servers are able to acquire more bandwidth, they can not keep up with the rapidly increasing requests of the users. Several systems appeared that alleviate the server from the dissemination process. The evolution of Peer-to-Peer systems gave a new way of attacking this problem. But as they got increasingly widespread, certain patterns became apparent. Uneven flow of data and intersperse congestion points compromised interdomain quality of service. In this paper we demonstrate how traditional systems meet users' demands, and present simulation results of a peer-to-peer approach based on a mobile agent platform. Our focus is on using the server's and the clients' resources in an optimum way to serve as many clients as possible in a certain period of time.

1 Introduction

When any sizable data 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], as that's the time new software are made available, in order to avoid congestion.

As today's needs for data transfer steadily increases, traditional ways of making data available to the masses become obsolete. Conventional HTTP and FTP servers can no longer serve as a way of distributing large amounts of data. That's because, in the case that no failures occur, the sum of the served clients could theoretically reach:

$$N = (\text{Server\_Bandwidth}) * (\text{Total\_Time}) / (\text{File\_Size})$$

Mirroring the required content on several disperse servers, doesn't always compensate for the rapid traffic increase, because for example modern Linux distributions can span more than one CD. Assuming for example 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.

The main architectural used for casting data through the Internet is IP multicast, which mainly targets realtime non-reliable applications. 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 recipients receive 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 keeping status of the download process for each client and managing downloads requests by users connected with different bandwidths. Forward Error Correction (FEC) has long being used for the dissemination of static data as it provides graceful degradation of performance in the presence of packet losses. It is based on (*n,k*) erasure code which encodes *k* source data packets into *n* encoded data packets, where $n>k$. The encoding guaranties that any given *k* of the *n* encoded packets are enough to reconstruct the *k* source data packets. FEC has been used in many multicast scenarios such as Filecast [1] and SwarmCast. 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 loss. 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 Peer-

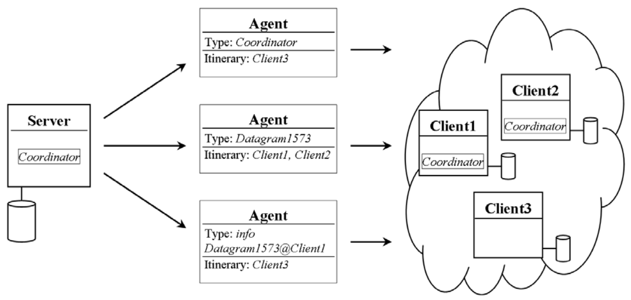

Figure 1. Mobile agent based dissemination paradigm

To-Peer application. 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. 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 farther interdomain jamming.

File sharing networks had never been designed for file dissemination. Nevertheless people turn to them to find highly anticipated software or even video file, 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 show how such files can be shared by these networks, and present a network that is designed to assist in file dissemination.

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 are summarized in section 5 and finally, section 6 briefly conceptualizes on the advantages of using mobile agents for content dissemination, and presents suggestions for further research.

2. Background

Software agents are programs that act on behalf of people. They are able to perform specified tasks that are assigned to them and can accomplish that with or without the supervision of the user, according to the given job.

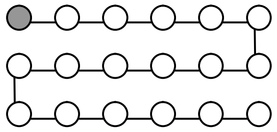
Figure 2. Pipelined clients

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 implements 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].

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 medium of 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, load them onto mobile agents and releasing them to the peers where the payload is delivered and continue according to their dynamic itinerary (figure 1). The coordination and communication overhead is acceptable considering the scalability that can be gained by the dynamic nature of the agents. 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 for 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. Conceptually, the easiest way to avoid retransmissions is to pipeline the file through all the

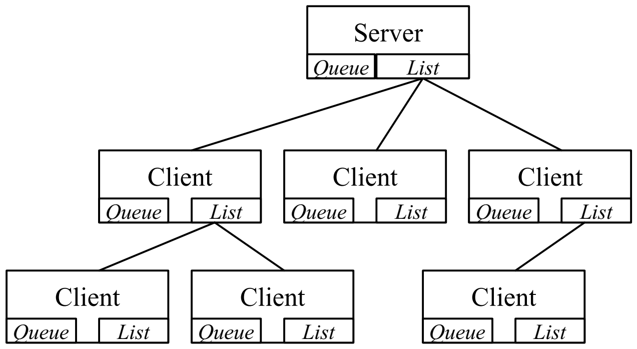

Figure 3. The dissemination network

clients (figure 2). 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 3). The server can upload the file to a certain amount 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 in 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 clients that it served (up to 10 entries) to client A, so that it can continue searching. In case that 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?

Another known issue is that clients with smaller speeds tend to stay on-line less time after they finish the download. That's because most of the time these clients don't have a permanent connection to the network (PSTN or ISDN users), or they need the bandwidth for something else, so they do not allow other users to download from them. Users going off-line leave the dissemination tree in an inconsistent state. Therefore it would be logical to serve the high-bandwidth clients first, as they are more likely to stay on the network for longer period of time, and assist new clients. What affects will that have in the long run?

4. Simulation model

In this section we present details of the simulation model for the proposed network, and show how different strategies might affect the dissemination process. The system was populated with clients arriving according to the exponential distribution using linearly changing mean interarrival time from 0.5 to 3.5 sec in a period of 3 days (simulated period: 259200 seconds) simulating this way a highly anticipated file. The file's size was set to be 650MB (the size of a full CD). The clients were separated in eight categories according to their bandwidth as shown in table 1.

Table 1. Clients' characteristics. Kbps and Mbps refer to kilobits per second and megabits per second.

Bandwidth	Percentage of total arrivals	Average time in the system after download is completed (in seconds)
64 Kbps	5%	3600
128 Kbps	5%	3600
256 Kbps	15%	14400
512 Kbps	15%	14400
768 Kbps	15%	14400
1 Mbps	15%	86400
10 Mbps	15%	259200
100 Mbps	15%	259200

As it was explained earlier, we accepted that clients connected to the net with higher bandwidth, stay on-line more time. This period of time is shown on Table 1, and it varied with exponential distribution in order to simulate network or client failure. Additionally, dial-up and ISDN clients are less likely to download such big files from the net. Therefore, we only allowed 10% of the clients to have bandwidths of 64Kbps and 128Kbps.

The server was set to have 100Mbps connection to the net and to never go offline. It has **no queue**, but it has a list of served clients of 100 entries. In order to have the server working at maximum speed, clients with bandwidths other than 100Mbps are not served by the server. 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. The clients have a queue of 10 entries and a list (of clients served by this client) of 20 entries. If a new client cannot be served or queued immediately, it waits for 300 seconds and retries.

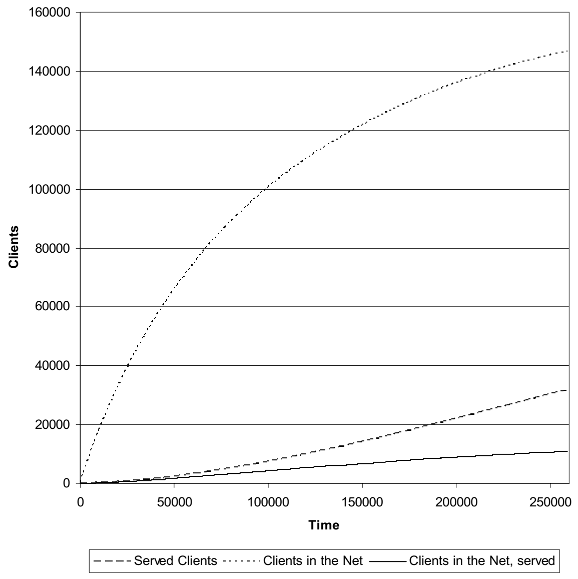

Figure 4. State diagram (Case 1)

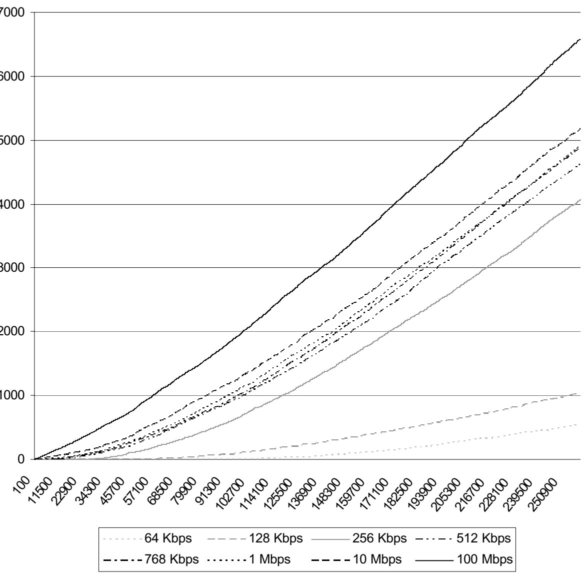

Figure 5. Served clients (Case 1)

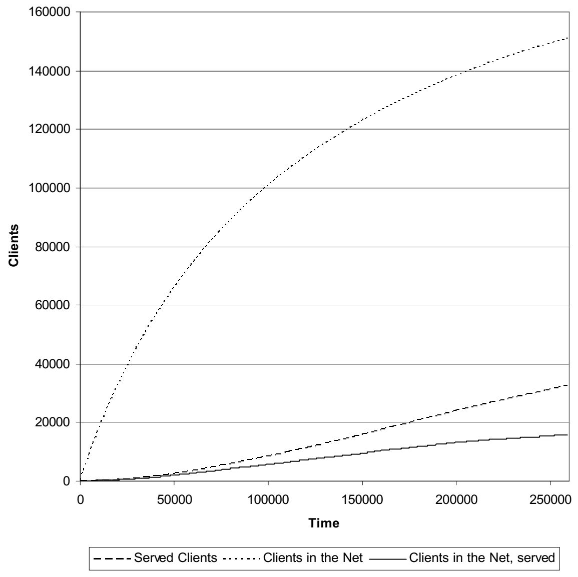

Figure 6. State diagram (Case 2)

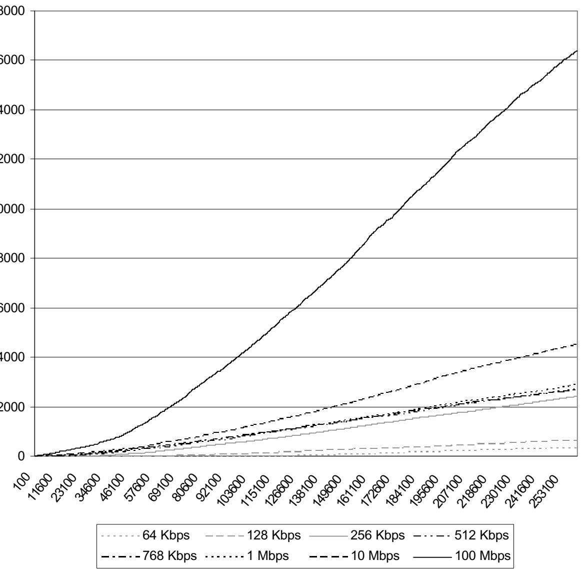

Figure 7. Served clients (Case 2)

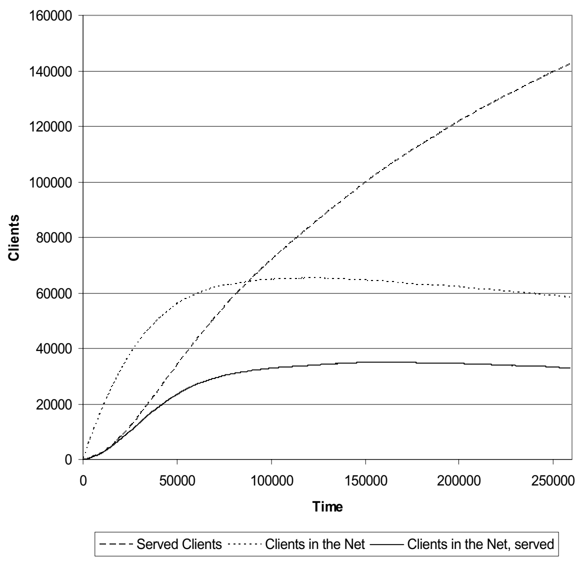

Figure 8. State diagram (Case 3)

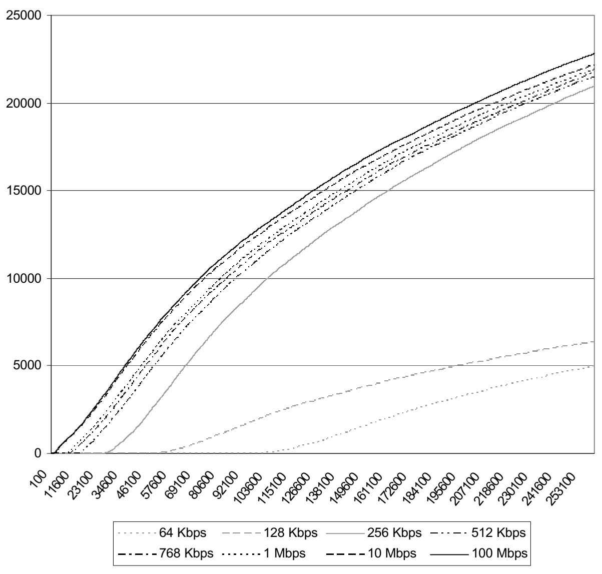

Figure 9. Served clients (Case 3)

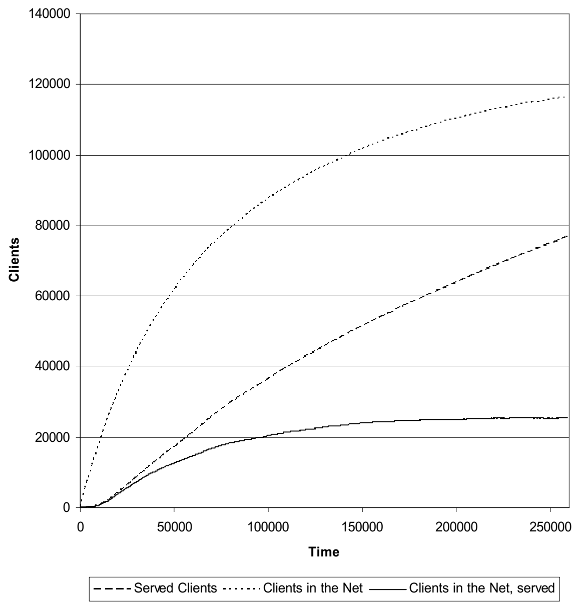

Figure 10. State diagram (Case 4)

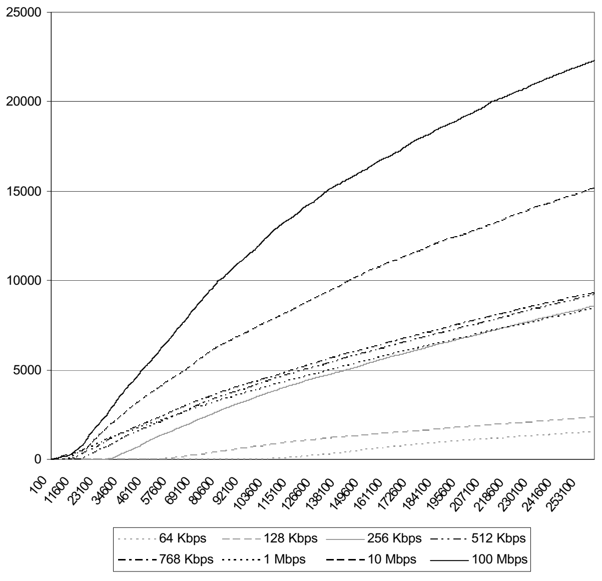

Figure 11. Served clients (Case 4)

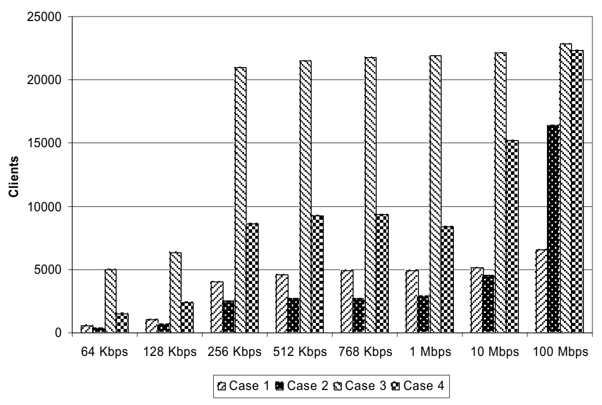

Figure 12. Number of clients that finished the download (up to the time the simulation ended)

As it was mentioned earlier, the behavior of this network can change dramatically under certain conditions. We investigate the system's performance in the following cases:

Case 1: A client may serve only one peer at a time. Therefore no multiple uploads are allowed. This is expected to decrease the mean service time.

Case 2: As in case 1, no multiple uploads are allowed. Additionally, the serving clients are allowed to queue only 2 peers (the queue size is 10) that have bandwidth less than that of the serving client. This way, clients use their bandwidth to the maximum and most of the clients are served by peers of similar speed.

Case 3: In this scenario, high-speed clients (1 Mbps, 10 Mbps and 100 Mbps) can serve up to 5 peers.

Case 4: Again up to 5 peers can be served simultaneously by a high-speed client, but as in case 2, a policy exists so that only 2 out of 10 peers served by each client, have inferior bandwidth than that of the serving client.

5 Results / Conclusion

As it is shown in tables 2 and 3, in the first case the mean time in queue is the biggest one from all, for speeds up to 1 Mbps. The mean service time of the 100 Mbps clients are also increased. This can be explained because no speed control policy is enforced. Therefore, high-speed clients can be forced to serve clients with much less bandwidth.

In the second case the best mean service time is observed for 100 Mbps clients. That's because the enforced policy makes good use of each client's bandwidth. However, this is not the case for the rest of the clients, which have similar service times to that of the clients in case 1. This can also be observed by comparing figures 5 and 7. While there are more than twice 100Mbps served client in case 2, the served clients on the rest of the categories decreased. The mean time in queue has also improved as it was expected. That's because 8 out of 10 clients in the queue have the same bandwidth as the serving client. Overall, as it is depicted in figure 4 and 6, the status of the network hasn't changed dramatically. That's because the increase of the number of 100Mbps served clients balances out the decrease on the rest of the categories. 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 high-speed clients in the beginning of the dissemination.

Table 2. Mean Response Time

	Case 1	Case 2	Case 3	Case 4
64 Kbps	167165	161317	126248	145192
128 Kbps	138986	124581	71865	103789
256 Kbps	120874	107210	43984	80802
512 Kbps	110416	100352	29306	68637
768 Kbps	110213	97230	24530	64296
1 Mbps	107208	97594	22185	65711
10 Mbps	104027	101775	14897	49491
100 Mbps	87178	70727	12825	20239

Table 3. Mean Service Time

	Case 1	Case 2	Case 3	Case 4
64 Kbps	104130	108394	114848	112974
128 Kbps	55046	53741	59979	61650
256 Kbps	28238	28069	31469	32640
512 Kbps	14323	14068	16770	17734
768 Kbps	9435	9631	11915	12737
1 Mbps	7537	7372	9512	12184
10 Mbps	1267	1447	2345	2224
100 Mbps	615	213	1393	251

Table 4. Number of clients that finished the download in 3 days.

	1st simulation	2nd simulation	Percentage
Case 1	29978	27679	7.67
Case 2	33118	30636	7.49
Case 3	141935	137281	3.28
Case 4	79177	77685	1.88

In the third case no speed policy was enforced, but the high-speed clients (and the server) were allowed to serve up to 5 clients simultaneously. This had negative affect on the mean service of all the clients. Especially 100 Mbps clients were served with up to 7 times slower speeds than that of case 2. That can be explained, since no measure has been taken to prevent a high-speed client to be queued on a lower speed client. Additionally, high-speed serving clients have to share their speed to up to 5 peers. On the other hand, in the third case we see the smallest mean time in queue for all clients and therefore, as seen in figure 12, the biggest amount of served clients. This is also shown in figure 9, where we can additionally observe that all the categories of client have similar number of served clients over time. That's due to the fact that the bandwidth of the serving clients is utilized to the maximum in most of the cases because of the multiple uploads. 64 and 128Kbps clients are shown to have significantly decreased numbers only because there was 5% of each of them in the system instead of 15% which was the case for the other categories.

As in the previous case, each client in the fourth case can serve up to 5 clients simultaneously. Additionally, the "2 out of 10" policy is enforced. Therefore we see mean service times equivalent to those in the second case, although slightly increased because the bandwidth is shared among several clients. However, the mean time in queue increased significantly, especially for the clients with limited bandwidth. As we can observe in figure 11, this has as a result the unfair treatment of the low-speed clients. That can be explained, as up to 5 same speed clients are assigned to a serving client of the same bandwidth. Therefore, although the serving client's bandwidth is utilized to the maximum, the clients being served use 1/5 of their bandwidth.

Overall, the mean time in queue is increased in all cases. But this is expected to decrease for simulation periods of several days. That's because, as more clients populate the dissemination network, the file is replicated to thousands of clients. Therefore a newly arriving client is more likely to find immediately another peer to be queued or served from. The steadily increasing mean service time as the bandwidth is getting lower, is expected. That's because clients with lower connections to the net, finish the download in longer periods of time. It should be noted that in the third case, as it can be seen in figure 8, a balance is reached in a relatively short period of time. This occurs when the number of departing clients matches the number of arriving clients. In spite of the rapid client arrivals in the beginning of the dissemination possess, the balance occurs in about a day. The fourth experiment offers an alternative for lower mean service times, but as shown in figure 10, the amount of served clients is significantly lower than that of the third experiment and the balance does not occur within the simulated time.

Furthermore, another experiment was implemented in order to check the grade of integrity of the system in each case. We run the four simulations again, but this time a 10% of the arriving clients were programmed to go off-line immediately after they get the file. By doing that, they do not assist at all at the dissemination process and therefore fewer clients are expected to finish the download in the given time of 3 days. Table 4 shows the consequences of this shortcoming.

The results reveal that the reduction of served clients is not grater than 10%. This shows that the algorithms do not degrade analogously, and especially case 4 proved to be more durable than the rest in this test. The fact that the observed decrease in the number of served clients is not as extensive as the number of clients denying to assist other clients, was expected because as the dissemination process continues, the increase of the served clients in the system becomes sufficient enough to counteract this problem. Table 4 also reveals that case 2 and 4 are more suitable for such scenarios than their counterparts. That can be explained as the dissemination tree maintained in those cases is more consistent over time. That's because low speed clients, which are not likely to stay for a long period of time in the system after they get the file, are placed at the lower levels of the dissemination tree.

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 128KB packets, to adjacent peers as they come, 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 are done in predetermine time intervals, called Epochs [13]. The peers are segmented in virtual groups according to their bandwidth and the epoch size is depended 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 issues raised in the third case. Additional we incorporated varying with time distributions as depicted in [14] for more realistic long-run simulations. We are also working towards creating a version that uses prior knowledge of a peer's content to push newly arrived packets and utilizes software FEC.

References

[1] Schooler E. & Gemmell J. (1997). Using Multicast FEC to solve the Midnight Madness Problem. *Microsoft research*.

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

[3] Parameswaran M., Susarla A. & Whinston A. (2001). P2P Networking: An Information-Sharing Alternative. *IEEE Computer*, v.34, pp. 31-38.

[4] Eytan A. & Huberman B. (2000). Free Riding on Gnutella. *Xerox Palo Alto Research Center*.

[5] Evangelos P. Markatos (2002). Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella. In the Proceedings of the CCGrid 2002: the second IEEE International Symposium on Cluster Computing and the Grid, May 2002, pages 65-74.

[6] Matei Ripeanu, Ian Foster, (2002). Mapping the Gnutella Network. IEEE Internet Computing, January 2002, pages 50-57

[7] Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., Tsudik, G., 1995. Itinerant Agents for Mobile Computing. Journal of IEEE Personal Communications, 2 (5).

[8] Harrison C., Chess D. & Kershenbaum A. (1995) Mobile Agents: Are They a Good Idea? Research report. IBM T.J. Watson Research Center, Yorktown Heights, New York.

[9] Joy B. (2000). Shift from Protocols to Agents. IEEE Internet Computing, v.4, pp.63-63.

[10] Spalink T., Hartman J. & Gibson G. (1999). 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, 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, Elsevier (Accepted for publication).

[12] K.G. Zerfiridis, and H.D. Karatza. "Mobile Agents as a Middleware for Data Dissemination”. Neural, Parallel & Scientific Computations, Dynamic Publishers, Atlanta, Vol 10, 2002, pp. 313-323.

[13] Karatza H. and Hilzer R.C. "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 (2002). "Task Scheduling Performance in Distributed Systems with Time Varying Workload”, Neural, Parallel & Scientific Computations, Dynamic Publishers, Atlanta, 10, 2002, pp. 325-338.