Fault Tolerant Peer-to-Peer Dissemination Network

Konstantinos G. Zerfiridis and Helen D. Karatza

Department of Informatics, Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
{zerf,karatza}@csd.auth.gr

Abstract. The widespread use of broadband networks gave new ground for Peer-to-Peer systems. The evolution of these systems made P2P file sharing networks one of the most popular ways of sharing content. Due to their distributed nature, such networks tend to be a reliable source for highly anticipated files. However, along with the benefits of P2P networks, certain patterns became apparent. Uneven flow of data and intersperse congestion points could result on increased mean response time or even network failure. In this paper the structure of Peercast, an agent based dissemination network, is presented. Emphasis is given to the self organizing nature of the dissemination tree, and simulation results depict its behavior under strenuous conditions.. . .

1 Introduction

While today the servers are able to acquire more bandwidth, they can not keep up with the rapidly increasing requests of the users. The demand for faster service increased as broadband connections became available. But if a file of considerable size has to be disseminated to a considerable amount of receivers, the network could 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 midnight madness problem [1].

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. 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

H. Kosch, L. Böszörményi, H. Hellwagner (Eds.): Euro-Par 2003, LNCS 2790, pp. 1257–1264, 2003.

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. IP multicast might be considered ideal for applications that require relatively high and constant throughput but not much delay. However 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 [2] are perhaps the most commonly used Peer-to-Peer applications. 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. Depending on the algorithm used, these sharing networks can be divided in two groups. Networks that maintain a single database of peers and their content references are known as centralized. Such file sharing networks [3] have several advantages, such as easy control and maintenance, and some disadvantages as, for example, server overload. On the other hand, dynamically reorganizing networks such as Gnutella [4], have a rather more elaborate service discovery mechanism, avoiding this way the use of a centralized server. Those kinds of networks are known as decentralized, and their main advantage is the absence of a single point of failure.

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. Extensive research has been done about how existing P2P networks operate over time and how they can be optimized [4,5]. However, the dissemination process of highly anticipated files on P2P networks over unreliable network connections remains unexplored. Peercast, a P2P network first presented in [6], is designed to assist the dissemination of a file in a heterogeneous network of clients. The purpose of this paper is to show Peercast's performance in the long term under different network conditions. Simulation results depict how network failures can affect this process.

The structure of this paper is as follows. In section 2 the Peercast's structure is shown, along with its latest extensions. Section 3 elaborates on the simulation model of the system and the simulated network failures. The results and drawn conclusions are summarized in section 4 and finally, section 5 presents suggestions for further research.

2 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 self-organizing tree of clients. The server 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 list (up to 100 entries). The server has a queue, but most of the clients are expected to find this queue full. This is the case especially at the beginning of the dissemination process, as clients arrive faster than the server can handle. In this case, 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 peer 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 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.

The peers are not expected to stay on-line for ever. But when a peer leaves the network, the dissemination tree is left in an inconsistent state. That's because the clients who were served from that peer are no longer accessible from the tree's root. In order to take advantage of all the peers that are willing to help in the dissemination process, the clients that are not occupied by serving other clients, periodically check with their parent peer. If the parent is not on-line or it is not accessible due to network failure, the client contacts the server and assigns itself as the server's child. If this fails, it requests from the server its list of served clients and tries to assigns itself to one of those clients. If this fails also, the client waits for a certain amount of time and retries.

As it was mentioned earlier, in order to avoid server explosion, the list of children that the server has, is limited. If this list is full and a new child has to be added, the server removes from the list its oldest child, accepts the new one, and forces the new peer to adopt the removed client to its own list. This way, the server always has a list of serving peers that recently became available and therefore are more likely to have empty queues and stay on-line longer.

In order to utilize all the available upload bandwidth, a single peer can serve several clients concurrently. Additionally, each client can initiate multiple concurrent download connections in order to utilize all the available download bandwidth. At the end of the transfer, the downloading client chooses randomly one of the assisting peers and requests to be listed on that one. The way to derive the optimal number of simultaneous upload connections and queue size is discussed in [6]. In brief, Peercast dynamically increases the number of connection slots used when the rate of arrivals balances with the rate of clients being served as shown in figure 1a. As a mean of keeping track of the number of served clients, 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 balance will occur. At that point it propagates a message to all the peers in its list and to every new client in order to increase the number of concurrent connections and decrease the queue size.

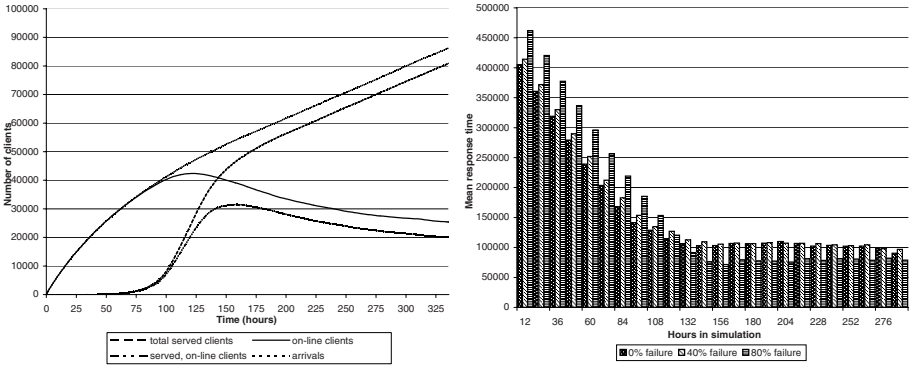

Fig. 1. a) Network's state over time, b) Mean response time in 12-hour intervals according to each client's arrival

Several issues arise about the performance of this algorithm under different network conditions in a heterogeneous network of clients. For example, how is the mean response time affected by several local congestion points or network failures? Can such problems affect dramatically the dissemination process? How does the system respond if a considerable number of peers refuse to assist?

3 Simulation Model

In this section details are presented about 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. 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%) had 384 Kbps and 128 Kbps, the third (20%) had 384 Kbps (download and upload), and the fourth (30%) 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 that file for ever. Therefore they were set to leave the dissemination network with exponential distribution and mean time of four days. The server (a client from category 4) was set to never go off line. An additional difference

between the server and the clients is that the server keeps a limited list of up to 100 clients that it served, whereas the clients have an unlimited list. That's because clients are not expected to stay on the network for ever, and therefore they do not run the risk of collapsing because of an overwhelming list.

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.

At the beginning of the dissemination, two download and two upload connections were used in order to speed up 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 (figure 1a). That happens when the rate of arriving and the rate of departing clients balance out. When the server estimates that the critical mass has been reached, it propagates a message through the peers in its list, notifying the clients to change the upload/download slots from 2 to 4. Additionally, the waiting queue on each client drops from 8 to 4 entries. The number of concurrent uploading and downloading streams at any time and the queue size are derived from simulation results showed in [6]. This switch is done in order to reduce mean response time, as newly arriving clients should not be queued on long queues. When the critical mass has been reached, they are more likely to find service from clients found deeper in the dissemination tree, optimizing this way all the available network's resources.

As it was mentioned earlier, the behavior of this network can change significantly under certain conditions. The system's performance is investigated at the beginning (2 weeks) of the dissemination, under different conditions. Our focus is on how the system behaves under network failure. More specifically, the simulations tested the system's performance when a certain percentage of connections failed. At any given time, the connection between any two clients (including the server) was set to have a predefined chance of failure. The system was tested using 0, 40 and 80 percent chance that any given client cannot contact another peer in order to be served or queued on it. This is done in order to simulate local network bottlenecks and network or system failures. If a client cannot contact a peer, it tries to find another. If no clients are found, it contacts the server to request an updated list of clients. In the case that even the server is not accessible, it retries after 600 seconds.

Additional simulations show the system's behavior when a percentage of the clients refuse assistance to other peers. In this case it is assumed that those clients go off-line immediately after they finish the download, and do not rejoin the network later on. This is expected to decrease dramatically the performance of the dissemination process. Nevertheless it is a behavior that can be expected. We test the system's performance when 0, 10 and 40 percent of the clients depart from the network immediately after they have been served. All of these simulations are done using a reliable network (0% connection failures). It should be noted that all network failure simulations were done using 10% of such clients.

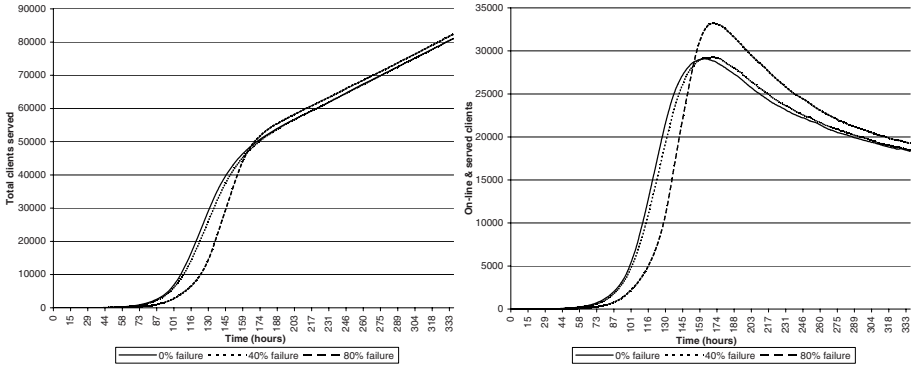

Fig. 2. Network failure tests over time: a) total clients served, b) served clients that are still on-line

The theoretical case that all the clients are willing to cooperate in the process for a certain period of time is examined in order to compare the degradation of performance that occurs to the rest of the cases.

4 Simulation Results and Conclusion

Figure 2a reveals that local network or system failures have an effect on the dissemination process, but not significant. More specifically, at the beginning of the dissemination there is a drop at the number of served clients. This is more obvious in the 80% network failure simulation. Nevertheless, as shown in table 1, the mean response time increased only by an average of 3.5% for the 40% case and 8.5% for the 80% case. This disproportional drop of performance is justifiable, as clients that are temporarily inaccessible by one peer, can be accessible by another. Figure 2a depicts that the network's resources are volatile at the beginning of the process, but as more peers join the network, their resource utilization increases steadily.

In figure 2b, the number of currently on-line peers is shown to be increased in the case of 80% network failure. That's because the server is not able to determine accurately when the critical mass has been built. This is also responsible for the higher mean response time shown in table 1. The increased mean response time in all cases can be explained as 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 1b where mean response time is shown in 12 hour intervals according to each client's arrival in the system.

An additional test that was executed at the end of the simulations showed the integrity of the dissemination tree under these conditions. More specifically, in the case of the 40% network failure, by iterating through the tree, 0.5% of the

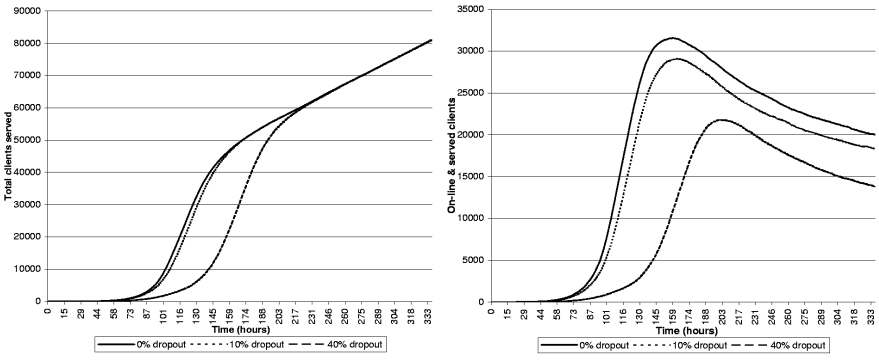

Fig. 3. Percentage of clients departs from the network immediately after completing the download a) total clients served over time, b) served clients that are still on-line

currently on-line serving-peers were found to be unreachable. This percentage increased to 10% for the 80% case. Those clients were in a timeout loop in order to try to reassign themselves in the tree. This shows that the nature of the self-organizing tree is relatively reliable to harsh conditions.

In figure 3a, the degradation of performance is shown to be proportional to the number of clients that depart from the system without assisting in the dissemination process. This is also shown in table 2 where the mean response time increased by an average of 4% when 10% of the clients leave the system as soon as they are served, and by 45% in the 40% case. Additionally, figure 3b shows that the critical mass is reached much earlier in the first case. That's because it is essential for the performance of the dissemination process that each client within this system acts as a server for other peers when it is served.

Further simulation results, not presented here due to space limitations, show that if a file of smaller size is used the impact of the clients that refuse to serve other peers is getting smaller. Overall, the system's behavior in strenuous conditions can be considered satisfactory. Its performance decrease is minimal in network failures, and the decentralized self-organizing tree of served clients proved to be reliable and scalable. The utilization of peer-to-peer technology for this task revealed a flexible way of reliably disseminating static data in a high number of clients, as long as the interested parties act collectively.

5 Future Work

Additional simulation experiments are under way, using distributions varying with time for more realistic long run simulations, as depicted in [7]. Peercast is an evolving platform. 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. A new version that replicates groups of 256KB packets, to adjacent peers as they arrive, is under way. This is expected to

Table 1. Mean response time over different amount of network failures (10% dropouts)

Net failures	256/256	384/128	384/384	1.5/384
0%	210377	199498	199672	182107
40%	216783	207291	207037	188381
80%	227610	216455	217156	197314

Table 2. Mean response time over different percentage of clients leaving immediately

Dropouts	256/256	384/128	384/384	1.5/384
0%	201812	192129	192077	174603
10%	210377	199498	199672	182107
40%	289458	278221	278129	257649

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 predetermine time intervals, called epochs [8]. 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.

References

1. Schooler, E., Gemmell, J.: Using Multicast FEC to solve the Midnight Madness Problem. Technical Report, Microsoft research (1997)
2. Parameswaran, M., Susarla, A., Whinston, A.B.: P2P Networking: An Information Sharing Alternative. IEEE Computer Journal, Vol. 34. (2001) 31-38.
3. Shirky C.: Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology / Listening to Napster. I.A. Oram (ed.), O'Reilly & Associates.
4. Ripeanu, M., Foster, I., Iamnitchi, A.: Mapping the Gnutella Network: Properties of large scale peer to peer systems and implications for system design. Internet Computing Journal, IEEE Computer Society (2002) 50–57
5. Markatos, E.P.: Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella. Proceedings of the CCGrid 2002, Second IEEE/ACM International Symposium on Cluster Computing and the Grid (2002) 65–74
6. Zerfiridis K.G., Karatza, H.D.: Large Scale Dissemination using a Peer-to-Peer Network. Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid 2003, Tokyo (2003) 421–427
7. Karatza, H.D.: Task Scheduling Performance in Distributed Systems with Time Varying Workload. Neural, Parallel & Scientific Computations, Vol. 10. Dynamic Publishers, Atlanta, (2002) 325–338
8. Karatza, H.D., 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 (2001) 36–42