CERIAS Tech Report 2002-20

ON DEMAND MEDIA STRAMING OVER THE INTERNET

by Mohamed M. Hefeeda, Bharat K. Bhargava, and David K. Y. Yau

Center for Education and Research in Information Assurance and Security, Purdue University, West Lafayette, IN 47907

On-Demand Media Streaming over the Internet*

Mohamed M. Hefeeda, Bharat K. Bhargava, and David K. Y. Yau
CERIAS and Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
{mhefeeda, bb, yau}@cs.purdue.edu

Abstract
Peer-to-peer (P2P) systems are gaining increasing attention in research as well as industrial communities. In this paper, we propose a new model for on-demand media streaming centered around the P2P paradigm. The proposed P2P model can support a large number of clients with a modest overall system cost. The P2P model allows for peers to *share* some of their resources with the system and in return, they get some *incentives* or rewards. We propose two architectures to realize (or deploy) the proposed model. One architecture relies on a special entity, an index, to facilitate locating the peers in the system. The second architecture is a pure P2P and builds an overlay layer over the participating peers. For each architecture, we present new *dispersion* algorithms (for disseminating the media files into the system) and *searching* algorithms (for locating peers with the required objects).

In addition, we conduct a cost-profit analysis of a media streaming service built on top of a P2P infrastructure. The analysis shows that with the appropriate incentives for the participating peers, the service provider achieves more profit. The analysis also shows how the service provider can maximize its revenue by controlling the amount of incentives offered to peers. By comparing the economics of the P2P and the conventional client/server media streaming architectures, we show that with a relatively small initial investment, the P2P model can realize a large-scale media streaming service.

Finally, we demonstrate the potential of the P2P model as an infrastructure for a large-scale on-demand media streaming service through an extensive simulation study on large, Internet-like, topologies. We evaluate several performance measures of the proposed model under different client arrival patterns such as constant rate arrivals, flash crowd arrivals, and Poisson arrivals.

1 Introduction

Streaming multimedia files to a large number of customers imposes a high load on the underlying network and the streaming server. The voluminous nature of the multimedia traffic along with its timing constraints make deploying a large-scale, cost effective, media streaming architecture over the current Internet a challenge.

The current media streaming architectures are mainly composed of a streaming entity and a set of requesting clients. The supplying entity could be one server, a set of servers, a set of servers and caches, or a set of servers and proxies. This entity is responsible for providing the requested media files to *all* clients. Figure 1 depicts a typical streaming architecture. The total number of concurrent clients the system can support, called the overall system capacity, is limited by the resources of the streaming entity. The limitation mainly comes from the out bound network bandwidth, but it could also be due to the processing power, memory size, or the I/O speed of the server machine. For instance, a streaming server hooked to the Internet through a T3 link ($\sim$

*This research is sponsored in part by the National Science Foundation grants CCR-001712 and CCR-001788, CERIAS, and IBM SUR grant

45 Mb/s) would be able to support up to 45 concurrent users requesting constant bit rate (CBR) media files recorded at 1 Mb/s. These approaches have limitations in reliability and scalability. The reliability concern arises from the fact that only one entity is feeding all clients, i.e., a single point of failure. The scalability of these approaches is not on a par with the requirements of a media distribution service that spans Internet-scale potential users, since adding more users requires adding a commensurate amount of resources to the supplying server. Throughout the rest of this paper, we generically refer to all variations of the aforementioned architectures as the *conventional* approach for media streaming.

Whereas deploying proxies and caches at several locations over the Internet increases the overall system capacity, it also multiplies the overall system cost and introduces many administrative challenges such as cache consistency and load balancing problems. The system's overall capacity is still limited by the aggregate resources of the caches and proxies. This shifts the bottleneck from one central point to a "few" distributed points, but it does not eliminate the bottleneck.

We propose a novel peer-to-peer media distribution model that can support a large number of clients with a modest overall system cost. The key idea of the model is that peers *share* some of their resources with the system and in return, they get some *incentives* or rewards from the service provider. As peers contribute resources into the system, the overall system capacity increases and more clients can be served. In addition, by properly motivating peers, the service provider can achieve a large system capacity with a relatively small initial investment. We believe that a peer-to-peer architecture has the potential to provide the desired large-scale media distribution service. We are motivated by the success of peer-to-peer file sharing systems such as Gnutella [18] and Napster [25]. Our architecture takes peer-to-peer file sharing systems a step further to provide a global media distribution service.

One important point worth clarification is the difference between a file-sharing system and a media streaming system [27]. In file-sharing systems, a client first downloads the *entire* file. Then it starts using the file. The shared files are typically small (few Mbytes) and thus take a relatively short time to download. A file is stored entirely by one peer and hence a requesting peer needs to establish only one connection to download it. Further, there are no timing constraints on downloading the fragments of the file, rather the total download time is more important. This means that the system (to some extent) is tolerable to inter-packet delays. In media streaming systems, on the other hand, a client *overlaps* downloading with consumption of the file, that is, it uses one part while downloading another to be used in the immediate future. The files are large (on the order of Gbytes) and take long time to stream. A large media file is expected to be stored by several peers, which requires the requesting peer to manage several connections concurrently. Finally, timing constraints are crucial to the streaming service, since a packet arriving after its scheduled play back time is useless and considered lost.

The main contributions of this paper can be summarized as follows. First, we propose a new P2P media streaming model, which is suitable for an on-demand media distribution service. Second, we present two architectures to realize (or deploy) the proposed model. One architecture relies on a special entity, an index, to facilitate locating the peers in the system. The second architecture is a pure P2P and builds an overlay layer over the participating peers. Third, we present new *dispersion* and *searching* algorithms for each architecture. The dispersion algorithms efficiently disseminate the newly published files into the system. The searching algorithms are for locating peers with the required objects. Fourth, we show the cost-effectiveness of the P2P model through a detailed cost-profit analysis. Fifth, we demonstrate the potential of the P2P model as an infrastructure for a large-scale on-demand media streaming service through an extensive simulation study on large, Internet-like, topologies. We evaluate several performance measures of the proposed model under different client arrival patterns such as constant rate arrivals, flash crowd arrivals, and Poisson arrivals.

The rest of the paper is organized as follows. Section 2 presents the P2P model. Section 3 presents the protocol to be run by a participating peer in the system. Two architectures to realize the P2P model and the associated searching and dispersion algorithms are presented in Section 4. Section 5 studies the economic issues of the P2P model. The simulation study is presented in Section 6. Section 7 summarizes the related

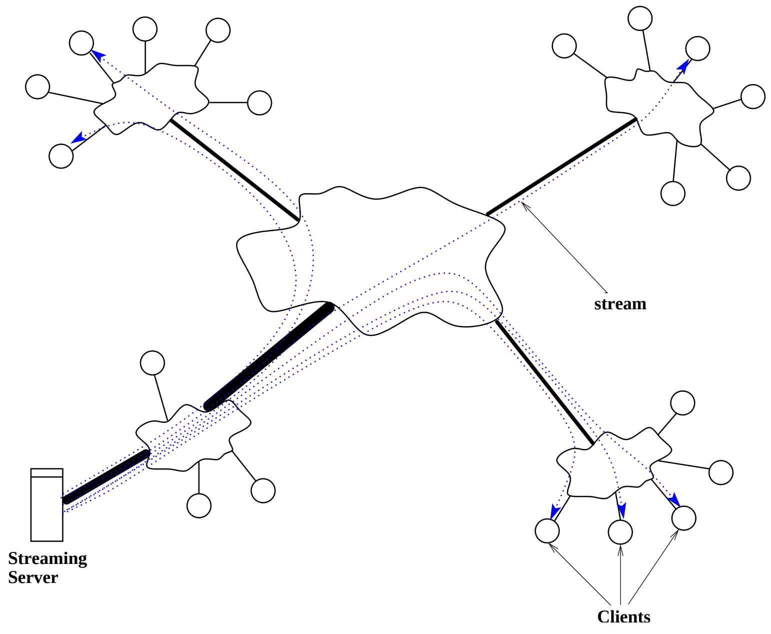


Figure 1: Abstract view of the current media streaming architectures. All clients are served by the streaming server.

research effort. Section 8 concludes the paper and proposes future extensions for this research.

2 P2P Model for Media Streaming

We now present our model for a large-scale media distribution service over the Internet. We first state the ultimate goal of the proposed model as follows:

To design an efficient and scalable architecture that satisfies the largest possible number of requests from the participating nodes with high quality of service and minimum overall system cost.

The basic idea of our approach is shown in Figure 2. In the P2P model, a peer may act as a client and/or as a mini-server. As a client, it requests media files from the system. A peer may opt to store segments of the media files that it has already consumed for a specific period of time. As a mini-server, it can provide these segments to other requesting peers in the system. We need to emphasize the miniature attribute of the mini-server, since the peer was never intended to function as a full server—serving many clients at the same time, rather, serving a few other peers for a limited time. Although, individually, each of these mini-servers adds only a little to the overall system capacity, combining a large number of them can significantly amplify the capacity of the system. Peers join the system along with their resources. Therefore, the more peers we have in the system, the larger the system capacity will be. This leads to a very scalable system that can potentially support an enormous number of clients.

The system as a whole benefits from the cooperative peers, which are willing to share their storage and bandwidth. Hence, a well designed peer-to-peer system should provide sufficient *incentives* to motivate peers to share their storage capacity as well as their network bandwidth. In a recent study of two popular peer-to-peer file sharing systems (Napster and Gnutella), Saroui *et al.* unsurprisingly discovered that peers tend to not share their resources with others without enough incentives [24]. The incentives may include, for example,

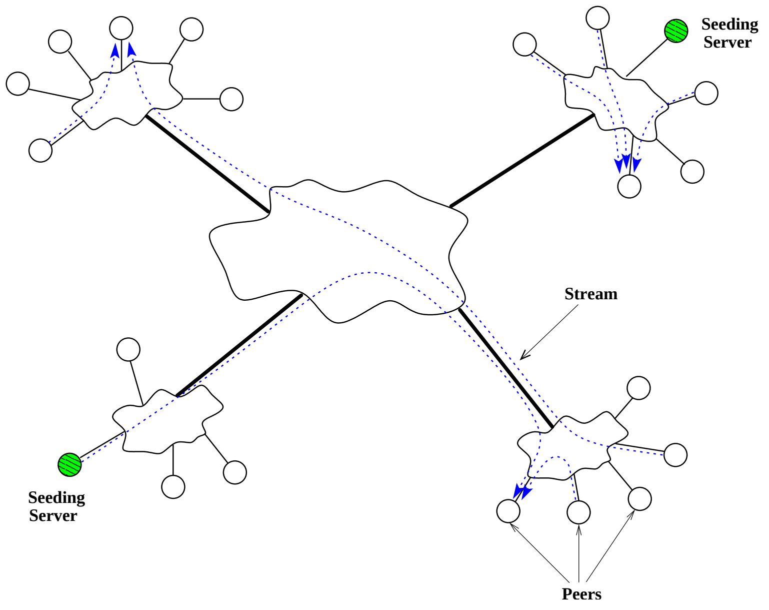


Figure 2: The proposed P2P architecture for media streaming. Peers help each other in streaming the requested media files.

lower rates ($/Byte) for those who store and supply media files to other peers in the system. Another way to encourage peers to share their resources is the "rewards for sharing" mechanism [8]. By this mechanism, points or credits are given to a cooperative peer as it shares more and more of its resources. Consuming peers, on the other hand, get penalized, by paying more to get the service, as they demand more resources from the system. Our cost-profit analysis, presented in Section 5, considers providing incentives to the cooperative peers and studies how these incentives affect the profit of the service provider.

2.1 The Model

As shown in Figure 2, our model consists mainly of a set of peers. In addition, we may have a set of seeding servers. These seeding servers provide or *seed* the newly published media files into the system. They stream these files to a limited number of peers, which in turn, will feed another larger set of peers, enlarging the system's capacity. After a reasonably short period of time, the system will have sufficient peers that already have the newly published media to satisfy almost all requests for the file without having to overload the seeding servers.

We formally define the entities involved in our model as well as their roles and how they interact with each other in the following.

1. **Peers.** This is a set of nodes currently participating in the system. Typically, these are machines of the clients who are interested in some of the media files offered by a streaming center. Let $\mathbb{P} = \{P_1, P_2, \cdots, P_{\mathcal{N}}\}$ be the set of all peers in the system. Every peer $P_i, 1 \leq i \leq \mathcal{N}$, specifies three parameters: (1) R_i (in Kb/s), the maximum rate peer P_i is willing to share with others; (2) G_i (in bytes), the maximum storage space the peer is willing to allocate to store segments of one or more media files; and (3) C_i, the maximum number of concurrent connections that can be opened to serve requesting peers. Recall that a peer is not meant to be a server, since it has limited resources. By using these three parameters, a peer has the ability to control its level of cooperation with other peers in the system.

2. **Seeding server.** In principle, one of the peers or a subset of them may seed the new media files into the system. However, in a *commercial* media streaming service, these seeding servers will typically be owned by a service provider. For the abstract model, it does not make any difference whether the seeding server is just a peer or a dedicated server. In contrast, the realization of the model, i.e., the deployable architecture, may differ if a dedicated server exists. Because this server may also be used to facilitate some crucial functions such as searching, dispersion, and accounting. We elaborate on this point in Section 4 when we describe the two different realizations of the model.

   We intentionally chose the name *seeding* (not streaming) servers to indicate that their main functionality is to *initiate* the streaming service and not to serve all clients at all times. These seeding "servers" do not negate the P2P nature of the model, in which peers help each other in getting the work done, i.e., streaming the media files.

3. **Stream.** A stream is a time-ordered sequence of packets belonging to a specific media file. This sequence of packets is not necessarily downloaded from the same serving node. Neither is it required to be downloaded in order. It must, however, be displayed by the client in order. It is the responsibility of the *scheduler* to download the packets from a set of possible nodes before their scheduled display time to guarantee non disruptive playing of the media.

4. **Media files.** The set of movies currently available in the system, or offered by the media center. Let $\mathbb{M} = \{M_1, M_2, \cdots, M_m\}$ be the set of all currently available movies in the system. Every movie has a size in bytes, and is recorded at a specific bit rate R Kb/s. We assume that R is a constant bit rate (CBR). A media file is divided into N segments. A segment is the minimum unit which a peer can cache. A supplying peer may provide the cached copy of the segment at a rate lower than the required rate R. Therefore, in general, one segment can be streamed to the requesting peer from multiple peers at the same time. According to our protocol (see Section 3), every peer will supply a different *piece* of the segment proportional to its streaming rate.

2.2 Advantages of the P2P Model

The P2P media streaming model offers several advantages over the conventional client/server model. It has the following desirable properties:

- **Scalability.** The architecture is highly scalable and we believe that it is a good candidate for an Internet-scale media distribution service. The scalability of the architecture is a result of two main reasons. First, the seeding server resources (e.g., storage capacity, network connectivity, and processing power) do not have to be in proportion to the number of nodes in the system, since peers help in providing the service. Besides, the reliance on the seeding server resources is diminishing as more peers join the system, and potentially, the server role as a media distributor will almost disappear. Network bandwidth saving is the second reason for scalability. In the conventional model, all clients need to be satisfied from the same server. Consequently, to satisfy a request of a distant client (in terms of the number of network hops), the traffic has to pass through many links. This consumes a lot of bandwidth and adds more load to, probably, already congested routers in the core of the network. The P2P approach relieves the network from much of this load by allowing a peer to obtain parts of the requested media file from *nearby* peers in the system which opt to store those parts.

- **Cost effectiveness.** Cost is a crucial factor for the success of a large-scale media streaming service. It is equally important for both the supplier and the consumers of the service. The supplier can satisfy many clients without a *gigantic* server with extremely high bandwidth network connectivity and without requirring a wide deployment of many costly caches and proxies over the Internet. Clients get less costly service by sharing some of their extra, often under-utilized, resources such as bandwidth and storage.

- **Ease of deployment.** The model is readily deployable over the current Internet, since it does not dictate any changes in the core of the network. In fact, it does not ask the network infrastructure for any support. The whole model can be realized at the application level.
- **Robustness.** Minimizing the dependence on the streaming servers enhances the overall system reliability. Recall that we have potentially many peers providing the same files, leading to a large degree of redundancy.

2.3 Challenges for the P2P Model

While the P2P model solves the scalability and reliability problems, it introduces new issues that need to be resolved. We summarize these issues in the following and propose solutions for some of them in the remainder of the paper.

- **Searching.** In the conventional client/server approach, the client knows the source from which to stream the entire movie. In the P2P approach, on the other hand, the client needs to search for other peers to get possibly the entire movie from them.
- **Scheduling.** The searching protocol presents a set of candidate nodes from which the movie can be downloaded. The search results contain some information about these nodes, e.g., the rate of each node and the segments of the movie that it stores. Now, it is the responsibility of the scheduler to decide which piece, from which node, and when to get in order to provide a continuous play back.
- **Dispersion.** We mentioned that each node may cache some segments of the movie that it has consumed before. Which segments to store? How many segments should a peer store? For how long? These questions need to be answered by the *dispersion* algorithm. The objective of a dispersion algorithm is that, a requesting peer finds most of the segments of the desired file in its "locality." This results in a reduced load on the network, shorter and less variable delays, and less reliance on the seeding servers.
- **Robustness.** For a P2P system to be robust, we need to consider two reliability issues: (1) Node failure: What if a node that is supposed to feed a segment of the movie fails (e.g., crashes due to a power outage, deliberately goes off line, deletes the stored pieces, or currently uses most of its bandwidth)? First, how to detect a failure? Second, how to react to it? (2) Network fluctuations: How to deal with fluctuations in the network, given that the state of the network is highly dynamic?
- **Security.** The objective of our model is to provide a high quality streaming service, which results in a profit for the service provider. Therefore, we need to deal with a malicious peer that intentionally tries to reduce the quality. That peer may, for instance, respond to a query and claim that it has segments of the movie. When the serving time comes, it denies providing the request, leaving the scheduler of the requesting client in a critical situation. Worse yet, it may feed the requesting client bogus packets—there is no way to check the packets before they actually played—leading to a poor quality of display. Another security aspect is how to provide the service only to the legitimate (paying) customers and deny it to those that roam around trying to get the service for free.

In this paper we address the first three challenges. We will consider the robustness and the security issues in the future work.

3 P2P Streaming Protocol

In this section, we describe the building blocks of the protocol used by a participating peer in the system. As shown in Figure 4, the protocol is composed of three phases and is to be run by a peer requesting a specific media file. In phase I, the requesting peer checks for the availability of the desired media file in the system. The phase starts with a crucial *searching* step. The employed searching mechanism depends on how the overall system is built, i.e., how peers are organized to form the system. We propose two main architectures: index-based and overlay. We describe these architectures and the searching techniques that can be used with them in Section 4.

The information returned by the searching step is arranged into a two-dimensional table. Each row j of the table contains all peers that are currently caching segment s_j of the requested file. Some information about each peer is also stored; e.g., its IP address, the available streaming rate, and some reliability information from the peer's history. Each row is then sorted to choose the most suitable peers to stream from. Several criteria can be used for sorting, such as proximity to the client (in terms of network hops), available streaming rate, and peer's average on-line time. A weighted sum of some (or all) criteria could also be used. In our experiments, we use the proximity as the sorting criterion. This reduces the load on the network, since traffic will traverse fewer domains. In addition, the delay is expected to be shorter and less variable, i.e., smaller jitter. Phase I ends with a verification step to make sure that all segments are available either solely from other peers or from peers and seeding servers as well. Otherwise, the requesting client backs off and tries later after exponentially increasing the waiting time.

The streaming phase starts only if phase I successfully finds all segments. Phase II streams segment by segment. It overlaps the streaming of one segment with the consumption of the previous segment. The playback of the media file starts right after getting the first segment. Because of the variability in network and peer conditions, buffering few segments ahead would result in a better playback of the media file. The buffering time can hide transient extra delays in packet arrivals. In the case that one of the supplying peers fails or goes off line, this buffering time may hide delyas due to finding and connecting to another peer from the standby table.

For every segment s_j, the protocol concurrently connects to all peers that are scheduled to provide pieces of that segment. The connections remain for time δ, which is the time to stream the whole segment. Different non-overlapping pieces of the segment are brought from different peers and put together after they all arrive. The size of each piece is proportional to the rate of its supplying peer. Let us define $\mathbb{P}^j$ as the set of peers supplying segment j. If a peer $P_x \in \mathbb{P}^j$ has a rate $R_x \leq R$, it will provide $|s_j|(R_x/R)$ bytes starting at wherever peer P_{x-1} ends. Since every peer supplies a different piece of the segment and $\sum_{x=1}^{|\mathbb{P}^j|} |s_j|(R_x/R) \geq |s_j|$, all pieces of the segment will be downloaded by the end of the δ period. To illustrate, Figure 3 shows three peers P_1, P_2, P_3 with rates $R/4, R/2, R/4$, respectively. The three peers are simultaneously serving different pieces of the same segment (of size 1024 bytes) to peer P_4.

Finally, in phase III, the peer may be allowed to cache some segments. This depends on the dispersion algorithm used. We present dipersion algorithms in Section 4.

4 Architecture

Two approaches are proposed to realize the P2P streaming service model described in the previous sections. Both follow the P2P paradigm, in which peers help each other in providing the *streaming* service. However, the two approaches are quite different in handling the *preparatory* steps of the streaming phase. The most important of these steps are: locating peers with the required media file (*searching*), and quickly disseminating media files into the system (*dispersion*). The chief distinction stems from the existence and the role of the *seeding* entity.

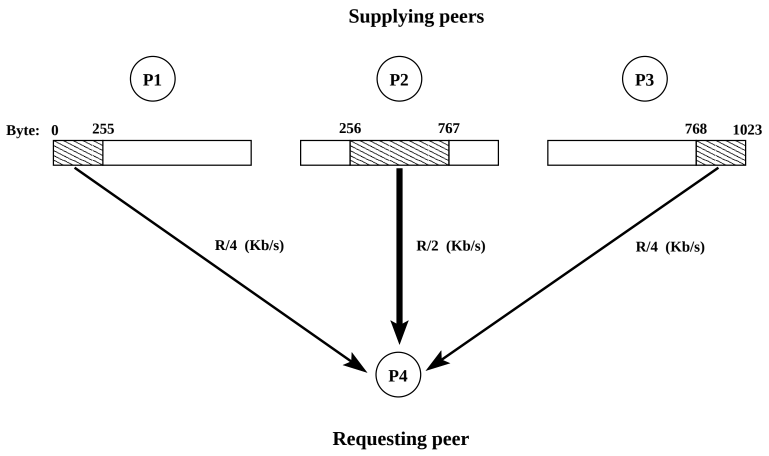


Figure 3: Peers P_1, P_2, and P_3 serving different pieces of the same segment to peer P_4 with different rates.

```
Protocol P2PStream
Phase I: Build AvailabilityTable (who has what)
a. Search for peers that have segments of the requested media file
b. Arrange the collected data in a two-dimensional table
/* Row $j$ contains the set of peers willing to provide segment $s_j$ */
let $\mathbb{P}^j = \{P_x | P_x \in \mathbb{P}$ and $P_x$ is listed in row $j\}$
c. Sort every row of the table in ascending order /* Based on the proximity */
d. Verify the availability of all segments with the full rate as follows:
for $j = 1$ to $N$ do
    if $\sum_{P_x \in \mathbb{P}^j} R_x \geq R$ then
        /* All pieces of segment $s_j$ are available */
        Choose "sufficient" peers to provide the required $R$
        Put the rest in a "stand by" table. /* might be used if a peer fails */
    else if one of the seeding servers can provide the deficit for $s_j$ then
        Add one more entry to row $j$ with $(seedServerID, R - \sum_{P_x \in \mathbb{P}} R_x)$
    else /* all seeding servers are busy */
        Back off, wait expo increasing time after every failed trial
    end if
end for
Phase II: Streaming
let $\delta$ = time to stream a segment
let $t_1 = 0$
let $t_j = t_{j-1} + \delta, j \geq 2$
for $j = 1$ to $N$ do
    At time $t_j$, get segment $s_j$ as follows:
    let $b_0 = 0$
    for all $P_x \in \mathbb{P}^j, 1 \leq x \leq |\mathbb{P}^j|$ do /* in parallel */
        Connect to $P_x$
        let $b_x = |s_j| \frac{R_x}{R}$
        Download the piece of segment $s_j$ from byte $b_{x-1}$ to byte $b_{x-1} + b_x - 1$
    end for
end for
Phase III: Caching
Store some segments (determined by the dispersion algorithm and the peer's level of cooperation)
```

Figure 4: The protocol used by a peer requesting a media file.

The first approach relies on having a special entity to maintain information about the currently participating peers. We call it the *index* approach. If the seeding entity is a set of servers owned by a provider, the index will typically be maintained by this set of servers. The second approach does not assign special roles to any peer, but needs to logically interconnect peers in the system, which we call the *overlay* approach. The following subsections describe the two approaches and present searching and dispersion algorithms that fit each of them.

Before we present the two approaches, we describe the *client clustering* idea, which is a key issue for both approaches. A cluster is defined as a logical grouping of clients that are topologically close to each other and likely to be within the same network domain [10]. It is highly beneficial, for both the client and the network, if a request can be fullfiled by peers within the same domain. For the network, it means that the traffic will travel fewer hops and hence will impose less load on the backbone links. The traffic delay will be shorter and less variable within the same domain, which is a desirable property for the streaming service.

We use a client clustering technique similar to the one proposed in [10]. The technique uses routing tables gathered from several core BGP routers. Client IP addresses that have the same longest prefix match with one of the routing table entries are assigned the same cluster ID. To illustrate the idea, consider five peers $P_1, P_2, P_3, P_4,$ and P_5, with IP addresses 128.10.3.60, 128.10.3.100, 128.10.7.22, 128.2.10.1 and 128.2.11.43, respectively. Suppose that among many entries in the routing tables, we have the following two entries: 128.10.0.0/16 and 128.2.0.0/16. The first three peers (all within Purdue University) share the same prefix of length 16 with the entry 128.10.0.0/16 (Purdue domain) and a prefix of length 12 with the entry 128.2.0.0/16 (CMU domain). Therefore, peers $P_1, P_2,$ and P_3 will be grouped together in one cluster with ID 128.10.0.0/16. Similarly, peers P_4 and P_5 will be grouped together in another cluster with ID 128.2.0.0/16. Notice that, using the same idea, a finer clustering within the same domain is also possible. For instance, P_1 and P_2 may be grouped in a smaller cluster with ID 128.10.3.0/24. This clustering technique does not incure much overhead, since it is performed once when the peer first joins the system.

4.1 Index Approach

Similar to Napster [25], this approach requires one (or a small subset) of the participants to maintain an *index* to all other peers in the system. The index can be maintained by the same machine seeding the media files (i.e., the seeding server), or by a separate machine. In any case, we call the maintainer of the index as the index server. This approach may be described as a *hybrid* scheme because the streaming process is peer-to-peer, while the searching and the dispersion processes are server-assisted. The main role of this special node is *not* to provide the streaming service, but to facilitate the searching and the dispersion processes. The load, in terms of CPU, bandwidth, and storage, imposed by the control information required by the searching and dispersion processes is a small fraction of the load imposed by the streaming service. To some extent, this alleviates the scalability and the single point of failure concerns that typically arise in such architectures. This approach greatly simplifies the searching process and reduces the overhead associated with it. Without the index, the overhead traffic puts a non-negligible load on the system. The index approach is *practically* easier and faster to deploy and more appropriate for a commercial media provider, since a commercial media provider would keep a server for accounting and charging customers and to *seed* the newly available media files into the system. We present a coarse-grained cost model for such systems in Section 5.

Finally, to enhance the scalability of the index approach, the index can be distributed over all clusters. In each cluster, one of the peers is *elected* to be the index server for the cluster. The clusters will be arranged in a hierarchical fashion, similar to the Domain Name System (DNS). This index organization needs protocols to elect the cluster's index server, and to keep the distributed index consistent and up to date. We are still inspecting the feasability of this idea.

```
Algorithm IndexSearch
/* Index server: upon receiving a query from peer $P_r$ */
c ← $getCluster(P_r)$
for $j = 1$ to $N$ do /* for every segment in the fi le */
    $candList[j]$ ← peers in c that have segment $s_j$
    if $\sum_{P_x \in candList[j]} R_x < R$ then
        if Peers from other clusters can provide the shortage then
            Append to $candList[j]$ suffi cient peers from the closest clusters
        else
            return empty list to $P_r$ /* $P_r$ backs off */
        end if
    end if
end for
return $candList$ to $P_r$
```

Figure 5: Index-based Searching algorithm

4.1.1 Index Searching

A key issue in the index approach is to keep the index current. First, notice that peers who are currently caching some of the media files are known to the index. Because they initially contact the index server to get served those media files. And, it is the index server that decides for them what to cache, as explained in the next subsection. Therefor, the index already knows who has what. The index server, though, does not know whether a peer is currently on or off line. Several techniques may be employed to keep the index up to date. In the case that a peer gracefully shutts down or reboots, a daemon running on the peer can send a notification message to the index server. Since it is unlikely that too many peers shut down synchronously, these notification messages will not cause message implosion at the index server. Another way to keep the index server current is to have the requesting client check the list of candidate peers returned by the index server by, for example, pinging them. The client then reports to the index server the status of all peers in the candidate list in one message.

The searching process is greatly simplified by the index server because it has a global information about all peers in the system. Figure 5 summarizes the searching process in the index approach. We assume that the index server gets the BGP routing tables and builds the clustering database a priori. Upon receiving a query from a client asking for a specific file, the index server first identifies the cluster to which the client belongs. If peers within the same cluster can satisfy the request, those peers will be returned to the client as a set of candiates to stream the request. Otherwise, peers from the closest clusters are chosen to serve the request. To find the closest clusters in terms of network hops, the same clustering idea can be applied *recursively*, that is, several smaller clusters are grouped together into a larger cluster if they share the same common network prefix. The index server, then, tries to satisfy the client's request from the larger cluster. For example, if we have peers P_1, P_2, P_3, P_4, and P_5, as described above, and P_1 is requesting a file. The index server will first try to satisfy the request from peers located within the cluster with ID 128.10.3.0/24, i.e., from peer P_2. If P_2 can not fullfill the request, the index server will try peers within the larger cluster with ID 128.10.0.0/16, i.e., from peers P_2 and P_3. If P_2 and P_3 can not fullfil the request, the index server will try to find peers from other clusters to make up the shortage. If the request can be fullfilled by any set of peers, this set is returned to the requesting client as a list of candidate peers. If the system does not have sufficient peers to satsify the request, an empty candidate peers list is sent to the client. The client then backs off and tries after an exponentially increased waiting time.

4.1.2 Index Dispersion

Caching the right segments of the media file at the right places is crucial to the incremental expansion of the system's capacity. The objective of the dispersion algorithm is to store enough copies of the media files

Table 1: Symbols used in the `IndexDisperse` algorithm.

Scope	Symbol	Description
System Variables	A	Average number of copies of the movie cached by all peers in the system
	Q	Average movie request rate in the system
Cluster Variables	$L_{\mathtt{c}}$	Next segment to cache in cluster c
	$a_{\mathtt{c}}$	Average number of copies of the movie cached by peers in cluster c
	$q_{\mathtt{c}}$	Movie request rate in cluster c
Peer Variables	N_x	Number of segments cached by peer P_x
	R_x	Rate at which peer P_x streams
	u_x	Fraction of time peer P_x is online
Movie Variables	N	Number of segments of the movie
	T	Duration of the movie (in hours)
	R	Rate at which the movie is recorded (CBR)

in each cluster to serve all expected client requests from that cluster. As described in Section 5, peers may need some *incentives* to cooperate; especially, if the service is provided by a commercial provider. These incentives are costs imposed on the provider. For this reason, it is important to keep just the required capacity in the system. To do so, we propose a dynamic dispersion algorithm that adjusts the capacity within each cluster according to the average number of client requests from that cluster.

The dispersion algorithm works in the following setting. At a specific instant of time, the system can serve a certain number of requests concurrently. At the same time, a client P_y sends a request to the system to get the media file. The client also declares its willingness to cache up to N_y segments to serve them to other clients with rate R_y in the future. The dispersion algorithm decides whether or not this peer should cache, and if so, which specific segments it should cache. The algorithm should ensure that, on the average, the same number of copies of each segment is cached, since all segments are equally important. To clarify, consider a file with only two segments. Keeping 90 copies of segment 1 and 10 copies of segment 2 means that we have effectively 10 copies of the media file available. In contrast, keeping 50 copies of each segment would result in 50 copies of the media file.

The `IndexDisperse` algorithm, shown in Figure 6, is to be run by the index server. Consider one media file with N segments, rate R Kb/s, and duration T hours. The algorithm requires the index server to maintain three types of information: per-peer information, per-cluster information, and per-system (or global) information. Table 1 summarizes the symbols used in the algorithm and their meaning.

For every peer P_x, the index server maintains: (1) N_x, the number of segments which are currently cached by P_x; (2) R_x, the rate at which P_x is willing to stream the cached segments; and (3) $u_x, 0 \leq u_x \leq 1$, the fraction of time P_x is online. Recall that the peer is not available all the time.

For every cluster c, the index server maintains the following: (1) $L_{\mathtt{c}}, 1 \leq L_{\mathtt{c}} \leq N$, the next segment to cache. (2) $q_{\mathtt{c}}$, the average request rate (per hour) the media file is being requested by clients from c. $q_{\mathtt{c}}$ represents the required capacity in the cluster c per hour. (3) $a_{\mathtt{c}}$, the average number of copies of the movie cached by peers in cluster c. c is computed from the following equation:

$$a_{\mathtt{c}} = \sum_{P_x \text{ in } \mathtt{c}} \frac{R_x}{R} \frac{N_x}{N} u_x. \tag{1}$$

The summation in Equation (1) computes the effective number of copies available in the cluster. It accounts for two facts: first, peers are not always online (through the term u_x), and second, peers do not cache all segments at the full rate (through the term $R_x N_x / RN$). Dividing $a_{\mathtt{c}}$ by T results in the number of requests that can be satisfied per hour, since every request takes T hours to stream. Hence, $(1/T)a_{\mathtt{c}}$ represents the available capacity in the cluster c per hour.

The index server maintains two global variables: (1) $A = \sum_{\mathtt{c}} a_{\mathtt{c}}$, the average number of copies of the movie cached by all peers in the system. (2) $Q = \sum_{\mathtt{c}} q_{\mathtt{c}}$, the average movie request rate in the system. Q and $(1/T)A$ represent the global required capacity and the global available capacity in the system, respectively.

```
Algorithm IndexDisperse
$L_{\mathtt{c}} \leftarrow 1, \forall \mathtt{c}$
while TRUE do
  Wait for a caching request
  /* Got request from peer $P_y$ to cache $N_y$ segments with rate $R_y$ */
  $\mathtt{c} \leftarrow getCluster(P_y)$ /* identify client's cluster */
  Compute $a_{\mathtt{c}}, q_{\mathtt{c}}, A, Q$
  if $q_{\mathtt{c}} > a_{\mathtt{c}}$ or $Q \gg (1/T)A$ then /* need to cache in round robin */
    if $(L_{\mathtt{c}} + N_y - 1) \leq N$ then
      $L_e = L_{\mathtt{c}} + N_y - 1$
    else
      $L_e = N_y - (N - L_{\mathtt{c}} + 1)$
    end if
    Peer $P_y$ caches from segment $L_{\mathtt{c}}$ to segment $L_e$
    $L_{\mathtt{c}} = L_e + 1$
  end if
end while
```

Figure 6: Index-based dispersion algorithm.

The algorithm proceeds as follows. Upon getting a request from peer P_y to cache N_y segments, the index server identifies the cluster $\mathtt{c}$ of the requesting peer. Then, it computes $a_{\mathtt{c}}$, $q_{\mathtt{c}}$, A, and Q[1]. The algorithm decides whether P_y caches based on the available and the required capacities in the cluster. If the demand is larger than the available capacity in the cluster, P_y is allowed to cache N_y segments in a *cluster-wide round robin* fashion. To clarify, suppose we have a 10-segment file. $L_{\mathtt{c}}$ is initially set to 1. If peer P_1 sends a request to cache 4 segments, it will cache segments 1, 2, 3, and 4. $L_{\mathtt{c}}$, the next segment to cache, is now set to 5. Then, peer P_2 sends a request to cache 7 segments. P_2 will cache segments 5, 6, 7, 8, 9, 10, and 1. $L_{\mathtt{c}}$ is updated to 2, and so on. This ensures that we do not over cache some segments and ignore others.

Furthermore, the `IndexDisperse` algorithm accounts for the case in which some clusters receive low request rates while others receive very high request rates in a short period. In this case, the global required capacity Q is likely to be much higher than the global available capacity $(1/T)A$, i.e., $Q \gg (1/T)A$. Therefore, even if the intra-cluster capacity is sufficient to serve all requests within the cluster, the peer is allowed to cache if $Q \gg (1/T)A$ in order to reduce the global shortage in the capacity. The operator $\gg$ used in comparison is relative and can be tuned experimentally.

4.2 Overlay Approach

In this subsection, we describe how our P2P media streaming service model can be built in a purely distributed fashion. The overlay architecture is more appropriate for a cooperative *non-commercial* media service, since no peer is distinguished from the others to charge or reward them. This differentiates the overlay approach from the index one. In the index approach, the index can charge or reward peers and hence, it is more appealing for a commercial service.

The participating peers in this approach form an abstract network (or an *overlay*) among themselves over the physical network. Neighboring peers in the overlay may be several hops apart in the physical network, but this is hidden in the overlay layer. In the literature, there are several ways of building such an overlay. The key determining factor is the employed searching protocol, which typically takes on most of the overhead needed by the system to function. Protocols such as CAN [21], Chord [26], Pastry [22], and Tapestry [29] guarantee that the requested object will be located efficiently, i.e., in logarithmic number of steps, if the object exists in the system. This is a desirable property for our system. However, these protocols, in one way or another,

[1]Computing these quantities is not necessarily performed for every request, especially if the request arrival rate is high. Rather, they can be updated periodically to reduce the computational overhead. Also, these quantities are *smoothed* averages, not instantaneous values.

```
Algorithm OverlayDisperse
/* Peer $P_y$ wants to cache $N_y$ with rate $R_y$ */
for $i = 1$ to $N$ do
    $dist[i].hop \leftarrow hops[i]$ /* $hops$ is computed during the streaming phase */
    $dist[i].seg \leftarrow i$
end for
Sort $dist$ in decreasing order /* based on the $hop$ field */
for $i = 1$ to $N_y$ do
    Cache $dist[i].seg$
end for
```

Figure 7: The dispersion algorithm used in the overlay architecture.

assign unique IDs to objects. The IDs are then mapped onto nodes in a way that facilitates locating these IDs when the system is queried about them. This *rigid* assignment requires knowing the exact ID of an object to locate it. Systems built on top of these protocols will likely lack the flexibility of searching using partial names or keywords [28].

If searching using keywords is essential to the service, we may use either the Gnutella's approach of the controlled flooding [18], or one of the more efficient techniques proposed in [28]. We do not describe these searching protocols any further, since our architecture can use any of them with slight adaptations. We need, however, a new dispersion algorithm, which we describe next.

4.2.1 Overlay Dispersion

We assume that the participating peers are self-motivated and will cache as much as they can for the success of the service. Thus, the question is not whether a peer should cache after it gets served the media file. Recall that there are no monetary incentives offered to the peers in this case. Therefore, caching more will not hurt the system. The relevant question is, which parts of the media file should be cached at each peer to achieve the goal of maintaining copies of the media file near to the clients.

We propose a decentralized dispersion algorithm, `OverlayDisperse`, to answer this question. The algorithm is to be run by each peer *independently*. The idea of the algorithm is simple: Peers cache segments that they get from far away sources more than they do for segments obtained from nearby sources. If peer P_y had to bring a segment from a distant peer, no other peer in the vicinity of P_y (e.g., within the same cluster) has this segment available. Otherwise, by our scheduling protocol (Section 3), P_y would have gotten the segment from that peer. This means that, the cluster in which P_y lies needs to cache this segment.

As shown in Figure 7, during the streaming phase, peer P_y computes the number of network hops traveled by each segment from the supplying peers to P_y. This is easy to implement by using the TTL field of the IP header. By knowing the initial value of the TTL field of the IP packets carrying the segment, P_y can compute the hop count as the difference between the initial TTL and the TTL of the received packets. If the segment comes from more than one supplying peers, a weighted sum of the hop count values is computed; $hops[j] = \sum_{z=1}^{l}(R_z/R)h_z$, where $P_1, P_2, \cdots, P_l$ are peers that supplied segment j, and $h_1, h_2, \cdots, h_l$ are the number of hops that packets coming from $P_1, P_2, \cdots, P_l$ traverse, respectively. The algorithm sorts the array $hops$ in descending order. Then, peer P_y caches N_y segments associated with the highest N_y hop count values. We evaluate the `OverlayDisperse` algorithm and compare it against a random dispersion algorithm in Section 6.

5 Cost-Profit Analysis

In this section, we study the economic issues of a media streaming service deployed over a P2P architecture. The objective is three fold. First, we show that by offering the appropriate incentives for the participating

Table 2: Symbols used in the cost-profit analysis.

Symbol	Description
A	Average Number of copies of the movie cached by peers
C	Total fi xed cost incurred by the provider (in \$)
F	Total profi t made by the provider (in \$)
h	Analysis period (in hours)
l	Number of customers
m	Maximum number of concurrent customers that can be served by the server
N_x	Number of segments cached by peer P_x
u_x	Fraction of time peer P_x is online
R_x	Rate at which peer P_x streams
T	Duration of the movie (in hours)
t	Time variable (in hours)
V	Total revenue (in \$)
v	Revenue per customer (in \$/customer)
α	Inducement factor
γ	Depreciation factor

peers, the media provider can gain more revenue. Second, we show that with a relatively small initial investment in the basic infrastructure, the P2P architecture can realize a large-scale media streaming service. Third, we verify the claim that the P2P architecture is more cost effective and more profitable than the conventional architecture.

The analysis assumes that the provider will deploy a small set of *seeding* servers with a limited capacity. We consider the case of one movie with a duration of T hours. For this movie we compute the revenue and the cost over a period of $h \geq T$ hours. Table 2 lists the symbols used in the analysis.

5.1 Incentives for Peers and Revenue for the Provider

We study how the cooperative peers can contribute to the total revenue of the provider, if they get the appropriate incentive. Assume that the provider earns a fixed amount of dollars v for each customer that successfully receives the entire movie, i.e., v is the revenue per customer.

In the conventional architecture, all clients receive the movie from the centralized server (or its proxies/caches, if any). Therefore, the *maximum* total revenue the provider can make is limited by the capacity of the server and its proxies. To compute the maximum total revenue, we assume sufficient customer requests that keep the server busy. We define m_{conv} to be the maximum number of concurrent customers that can be served by the server and its proxies. Since every streaming session lasts for T hours, the maximum number of customers that can be satisfied per hour is m_{conv}/T. Therefore, the maximum total revenue V_{conv} during the period h is given by:

$$V_{conv} = \frac{h}{T}\, m_{conv}\, v. \tag{2}$$

In the P2P architecture, clients get portions (possibly all) of the media file from other peers. This allows for serving more customers with the same server capacity. However, peers should get a part of the revenue. Otherwise, they would have no motivation to share their bandwidth and storage. Assume that the provider grants (to peers) αv dollars ($0 \leq \alpha < 1$) for every customer served by those peers in the system. We call α the *inducement factor*. A higher α will induce more cooperation from the peers. Every contributing peer gets a *share* of the revenue proportional to its contribution. For example, if peer P_x serves the entire request, i.e., provides all the N segments at the full rate R, it will get αv dollars. Whereas, if P_x serves only $N_x, 0 \leq N_x \leq N$, segments at rate $R_x, 0 \leq R_x \leq R$, P_x gets $(R_x/R)(N_x/N)\alpha v$ dollars.

As explained in Section 4.1.2, A is defined as the average number of copies cached by the set of of peers

$\mathbb{P}$ in the system and is computed as:

$$A = \sum_{x=1}^{|\mathbb{P}|} \frac{R_x}{R} \frac{N_x}{N} u_x. \tag{3}$$

Dividing A by T results in the number of requests that can be satisfied per hour, since every request takes T hours to stream. Multiplying the result by h gives the total additional number of customers l_{add} that can be satisfied by peers during the period h. Thus, l_{add} is computed as:

$$l_{add} = \frac{h}{T} A. \tag{4}$$

From the provider's perspective, l_{add} is the total number of additional customers (beyond the seeding servers capacity) that can be served due to contributions by peers. Each additional customer adds $(1 - \alpha)v$ dollars to the total revenue of the provider. We define m_{p2p} to be the maximum number of concurrent customers that can be served by the seeding servers (typically, $m_{p2p} \ll m_{conv}$). The maximum total revenue in the P2P architecture for the period h is given by:

$$V_{p2p} = \frac{h}{T} m_{p2p} v + l_{add} (1 - \alpha) v. \tag{5}$$

The maximum total revenue is not limited by the maximum capacity m_{p2p} of the seeding servers. Rather, it grows in proportion to l_{add} (since $\alpha < 1$).

Two questions arise: How do we get peers to cache A copies? How long will it take? Intuitively, A is positively correlated with the inducement factor α and the time t. Larger α values motivate either more peers to cache, or the same peers to cache more segments, or both. This increases the effective number of cached copies of the movie. For a given α, the system gets a specific caching rate from peers. Thus, the number of cached copies accumulates over the time. Generally speaking, A can be expressed as a function of α and t, i.e.,

$$A = f(\alpha, t). \tag{6}$$

The function f can be estimated (and refined over time) through statistics gathered from customers. It depends on how peers react to the incentives offered by the provider. To illustrate the analysis procedure, we consider a simple *hypothetical* function:

$$A = f(\alpha, t) = k \, \alpha^{\frac{1}{\epsilon_1}} \, t^{\frac{1}{\epsilon_2}} \tag{7}$$

where k, ϵ_1, and ϵ_2 are positive constants greater than one. Notice that, ϵ_1 and ϵ_2 are unitless. The unit of k should be chosen to balance the equation; it depends on the value of ϵ_2. These constants are used to control the shape of the function f.

We plot Equation (7) for $k = 100$, $\epsilon_1 = 2$, $\epsilon_2 = 3$, and several values of α in Figure 8. Equation (7) and Figure 8 show the expected positive correlation between A and both α and t. They also indicate that the number of cached copies A will saturate after a reasonably long time, which is intuitive because we have a finite number of peers with finite capacities. We believe that the shape of a realistic $f(\alpha, t)$ (i.e., one that is fitted to data gathered from customers over a long period of time) will look somewhat similar to the curves shown in Figure 8. We are unable to verify this conjuncture due to the lack of data.

Figure 8 can be used to answer the two questions as follows. Suppose that the provider expects a total of 30,000 client requests distributed over a period of 20 days. Assume a movie of a duration $T = 2$ hours and an average active period per day of 10 hours. Thus, $h = 20 \times 10 = 200$ hours. If we ignore the limited capacity of the seeding servers, the provider needs $l_{add} = 30,000$ customers to be served by peers. Thus, from Equation (4), A needs to be 300 by the end of the 200-hour period. In Figure 8 we see that the minimum value of α that can achieve the expected capacity within the 200-hour period is 0.3. Taking $\alpha = 0.1$ will take about 850 hours to get the required capacity. On the other hand, a 0.7 value for α will quickly motivate peers,

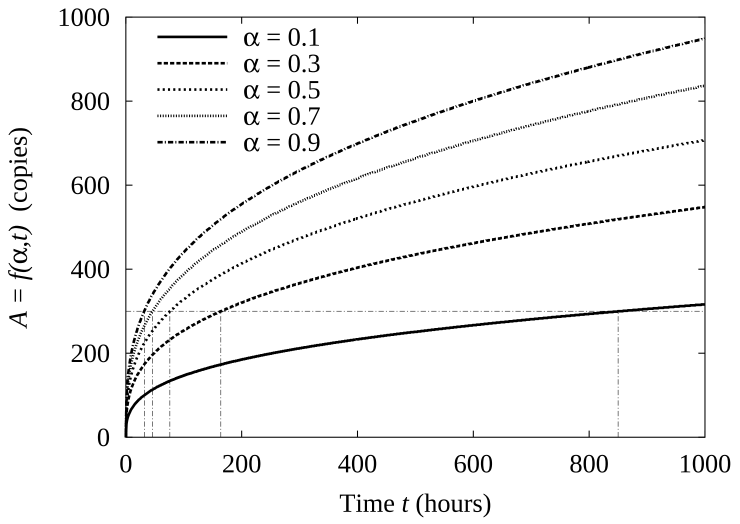


Figure 8: The effect of the inducement factor α on the average number of cached copies A over a period of time t.

and the required capacity will be obtained in less than 46 hours. However, as Equation (5) indicates, $\alpha = 0.7$ will yield a smaller revenue for the provider than $\alpha = 0.3$.

In summary, using a sample function $f(\alpha, t)$, we showed how the provider can determine the appropriate inducement factor to motivate peers to cache enough copies of the media file for the expected number of customers within a specific target period.

5.2 Revenue Maximization

Suppose that the provider expects many customers that will absorb any offered capacity in the system within a period of h hours. This can happen in the case of a release of a popular movie. Given the estimated $f(\alpha, t)$, we can find the optimal inducement factor α_{opt} that maximizes the total revenue for the provider. In this case, t is fixed and equals h, since we maximize the total revenue over the entire period h. If we substitute $f(\alpha, h)$ in Equation (5), we get the revenue V_{p2p} as a function of only one unknown, α. As an example, consider our hypothetical $f(\alpha, t)$ as described above. After substitution, Equation (5) reduces to:

$$V_{p2p} = \frac{h}{T}\, m_{p2p}\, v + k\, \alpha^{\frac{1}{\epsilon_1}}\, h^{\frac{1}{\epsilon_2}}\, \frac{h}{T}\, (1 - \alpha)\, v. \tag{8}$$

Differentiating (8) w.r.t. α and equating the result to zero, we get the optimal inducement factor α_{opt}:

$$\alpha_{opt} = \frac{1}{(1 + \epsilon_1)}. \tag{9}$$

Figure 9 shows the relation between V_{p2p} and α for $\epsilon_1 = 1, 2, 3,$ and 4. The figure shows the importance of choosing the correct inducement factor to maximize the revenue for the provider. For instance, if $\epsilon_1 = 2$, the optimal inducement factor α_{opt} is 0.33 and it achieves the optimal total revenue of \$18,865. In contrast, for $\alpha = 0.1$, the provider achieves a total revenue of \$14,210, and for $\alpha = 0.6$, achieves a total revenue of \$15,381.

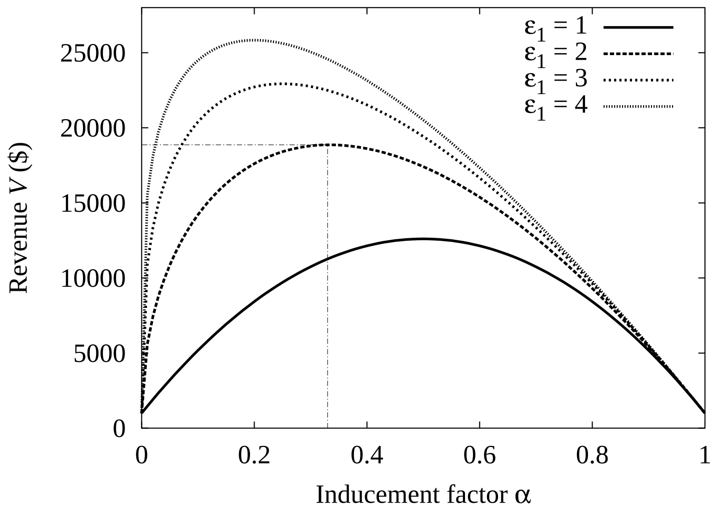


Figure 9: Finding the optimal α to maximize the provider's revenue

5.3 Profit Comparison

This section compares the cost and profit for the P2P architecture and the conventional architecture. The comparison is approximate and ignores several practical factors. Nevertheless, it gives us some guidelines for the feasability of deploying a large-scale media service deployed over a P2P infrastructure.

We first define the net profit made by a provider within a period of h hours as:

$$F = V - C, \tag{10}$$

where V is the total revenue obtained by serving customers during the period h, and C is the total cost incurred by the provider during that period. C considers only the fixed cost, because the running cost is implicitly considered in computing V. The running cost is considered in the revenue per customer v. The fixed cost is the cost for deploying the whole infrastructure. A reasonable assumption to make is that the fixed cost is proportional to the maximum number of customers that can be served concurrently by the server. To serve more customers, more powerful machines and larger network bandwidth are needed. Since we analyze the net profit over a period of h hours, we consider the cost over the same period. Thus, the total cost within a period of h hours (for both architectures) can be approximated as:

$$C = \gamma\, m \tag{11}$$

where γ is the proportional factor. γ also accounts for the fact that the cost incurred within the period h is a fraction of the total cost needed to deploy the infrastructure. We may call γ the *depreciation* factor, which is proportional to the decrease in the monetary value of the machines due to their use for h hours.

To compute the maximum net profit, we need to compute the total number of customers l that can be served during the period h. For the conventional architecture, customers can be satisfied as long as the server's capacity allows. We define l_{conv} as the maximum number of customers that can be served by the server in h hours. It is computed as:

$$l_{conv} = \frac{h}{T}\, m_{conv}. \tag{12}$$

The net profit as a function of the number of customers l for the conventional architecture can be computed as:

$$F_{conv} = \begin{cases} l\,v - C_{conv}, & 0 \le l \le l_{conv} \\ l_{conv}\,v - C_{conv}, & l > l_{conv} \end{cases} \tag{13}$$

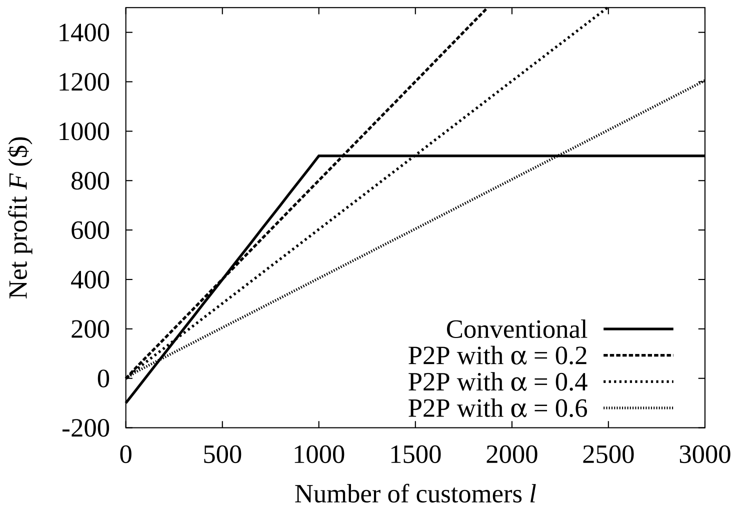


Figure 10: Profit comparison between conventional and P2P architectures.

Similarly, the maximum number of customers that can be served by the seeding server l_{p2p} in the P2P architecture is given by:

$$l_{p2p} = \frac{h}{T}\, m_{p2p}. \tag{14}$$

Besides these l_{p2p} customers, the P2P architecture can serve additional customers due to peers contributions. For each additional customer, the provider gains $(1-\alpha)v$ dollars. For simplicity, we assume that the additional capacity added by peers will be used only when the seeding server runs out of capacity. Therefore, the net profit for the P2P architecture can be computed as:

$$F_{p2p} = \begin{cases} l\, v - C_{p2p}, & 0 \le l \le l_{p2p} \\ l_{p2p}\, v - C_{p2p} + & \\ (l - l_{p2p})\,(1-\alpha)\, v, & l > l_{p2p} \end{cases} \tag{15}$$

To illustrate this comparison, let us consider the following values: $T = h = 1$, $v = 1$, $\gamma = 0.1$, $m_{conv} = 1000$, and $m_{p2p} = 0.01 \times m_{conv} = 10$ $(m_{p2p} \ll m_{conv})$. Hence, $l_{conv} = 1000$, $l_{p2p} = 10$, $C_{conv} = 0.1 \times 1000 = 100$, and $C_{p2p} = 0.1 \times 10 = 1$. Using these values, Equations (13) and (15) are reduced to become:

$$F_{conv} = \begin{cases} l - 100, & 0 \le l \le 1000 \\ 900, & l > 1000 \end{cases} \tag{16}$$

$$F_{p2p} = \begin{cases} l - 1, & 0 \le l \le 10 \\ 9 + (1-\alpha)\,(l - 10), & l > 10 \end{cases} \tag{17}$$

In Figure 10, we show the net profits for the conventional architecture and the P2P architecture with $\alpha = 0.2, 0.4$, and 0.6. The figure shows that with a much less investment in the basic infrastructure (one hundredth), the P2P architecture is able to achieve a comparable profit with a moderate number of customers. Moreover, the P2P architecture achieves much higher profit as the number of customers increases. Even when $\alpha = 0.6$ (meaning that peers are getting a larger portion of the per-customer profit than the provider), the P2P architecture still achieves a higher profit for the provider for $l \ge 2225$. Finally, the conventional architecture has negative profit (or loss) for $l < 100$. In contrast, the P2P architecture has negative profit for only $l < 1$. This is a desirable property, especially for small and start up companies.

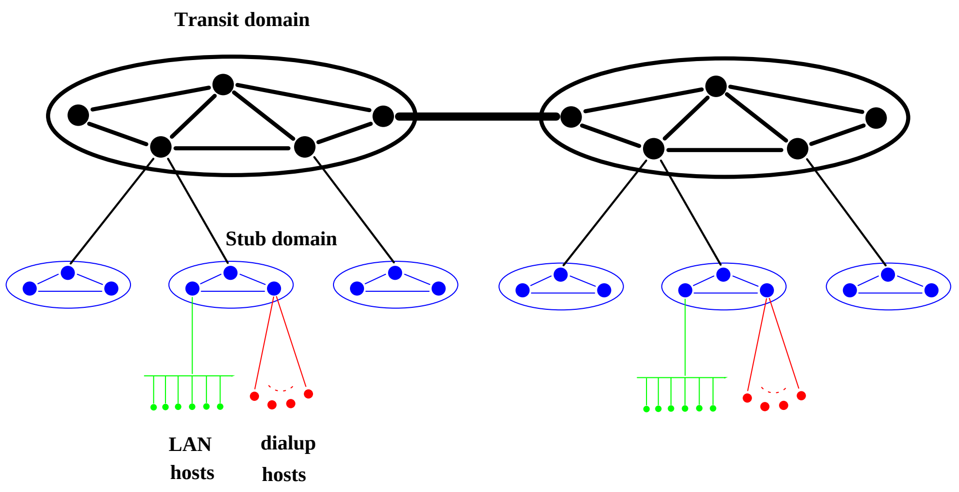


Figure 11: Part of the topology used in the simulation.

6 Evaluation

In this section, we evaluate the proposed P2P model through extensive simulation experiments. First, we study the performance of the P2P model under various situations, e.g., different client arrival patterns and different levels of cooperation offered by the peers. We are interested in the following performance measures as the system evolves over the time:

1. The overall system capacity, defined as the average number of clients that can be served *concurrently* per hour;
2. The average waiting time for a requesting peer before it starts playing back the media file;
3. The average number of satisfied (or rejected) requests; and
4. The load on the seeding server.

Second, we evaluate the proposed cluster-based dispersion algorithm and compare it against a random dispersion algorithm. The comparison criteria are: (1) the percentage of the requests satisfied within the same cluster, and (2) the reduction of the load on the underlying network due to careful dissemination of the media files over the participating peers.

We simulate a large (more than 13,000 nodes) hierarchical, Internet-like, topology. We use the GT-ITM tool [2] for generating the topology and the Network Simulator *ns-2* [19] in the simulation.

6.1 Performance of the P2P Architecture

6.1.1 Topology

Figure 11 shows a part of the topology used in the simulation. Approximately resembling the Internet, the topology has three levels. The highest level is composed of transit domains, which represent large Internet Service Providers (ISPs). Stub domains; which represent small ISPs, campus networks, moderate-size enterprise networks, and similar networks; are attached to the transit domains on the second level. Some links may exist among stub domains. At the lowest level, the end hosts (peers) are connected to the Internet through stub routers. The first two levels are generated using the GT-ITM tool [2]. We then, probabilistically add dialup and LAN hosts to routers in the stub domains.

The topology used in this set of experiments consists of 20 transit domains, 200 stub domains, 2,100 routers, and a total of 11,052 hosts distributed uniformly at random. More details about the topology generation as well as the simulation scripts and programs are available at [20].

6.1.2 Simulation Scenario

We simulate the following scenario. A seeding server with a limited capacity introduces a media file into the system. According to the simulated arrival pattern, a peer joins the system and requests the media file. Then, the `P2PStream` protocol, described in Section 3, is applied. We do not assess the overhead imposed by the searching step in this set of experiments. If the request can be satisfied, i.e., there is a sufficient capacity in the system, connections are established between the supplying peers and the requesting peer. Then, a streaming session begins. The connections are over UDP and carries CBR traffic. If the requesting peer does not find all the segments with the full rate, it backs off and tries again after an exponentially increased waiting time. If the waiting time reaches a specific threshold, the request is considered "rejected" and the peer does not try again.

When the streaming session is over, the requesting peer caches some of the segments depending on the level of cooperation, called the caching percentage. For instance, if the caching percentage is 10% and the media file has 20 segments, the peer stores two randomly-chosen segments. The peer also selects a rate at which it wants to stream the cached segments to other peers.

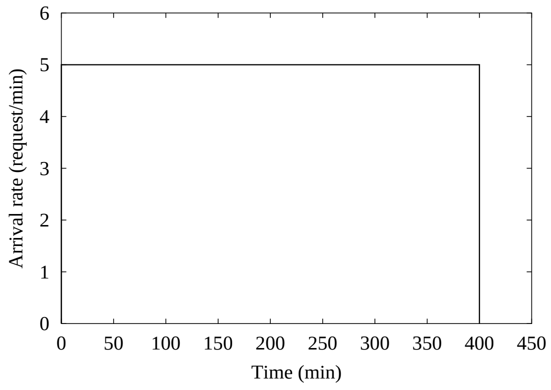
6.1.3 Simulation Parameters

We have the following fixed parameters[2]:

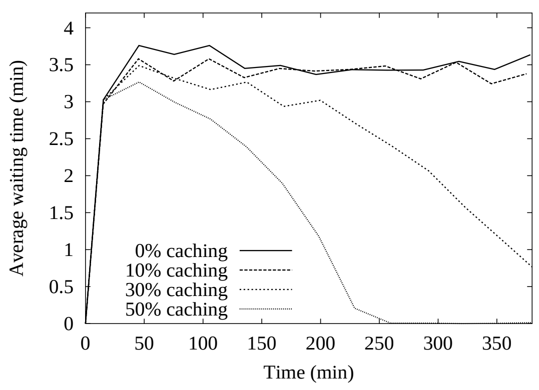
1. A media file of 20 minute duration recorded at a CBR rate of 100 Kb/s and divided into 20 one-minute segments;
2. The dialup peers are connected to the network through links with 1 Mb/s bandwidth and 10 ms propagation delay;
3. The LAN peers have 10 Mb/s Ethernet connectivity with a 1 ms propagation delay;
4. The backbone links have a bandwidth of 155 Mb/s with variable delays, depending on whether a link is between two routers in the same stub domain, the same transit domain, or across domains;
5. The seeding server has a T1 link with a bandwidth of 1.5 Mb/s, which means that it can support up to 15 concurrent clients;
6. The requesting peer can open up to 4 connections with other peers to get a segment at the desired rate of 100 Kb/s; and
7. The maximum waiting time for a requesting client is two minutes.

We vary the caching percentage from 0% to 50% and study the system under various client arrival patterns. The results are summarized in the follow subsections.

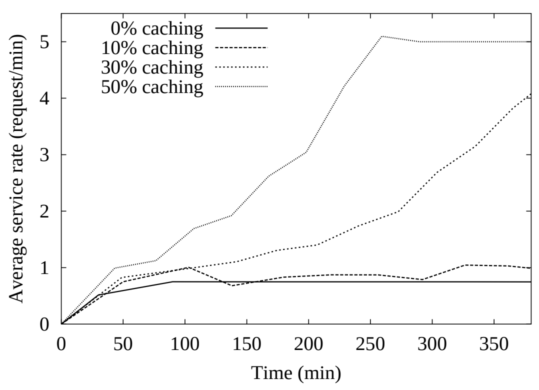
[2]When we used these parameters, the simulation time and the memory requirements were excessive. So, we scaled down all rates by a constant factor to accelerate the simulation. For example, instead of streaming at 100 Kb/s on a peer's link of 1 Mb/s and a backbone bandwidth of 155 Mb/s, we use the values 0.1 Kb/s, 1 Kb/s, and 155 Kb/s, respectively. Essentially, we put relatively the same stress on all components of the system, but we send much fewer packets in the simulation. Thus, we save a lot of memory and time. With this and even more aggressive scaling, each experiment lasted from 20 to 40 hours and required from 500 to 900 GBytes of memory on a 2 GHz Intel machine with a 512 GByte physical memory.



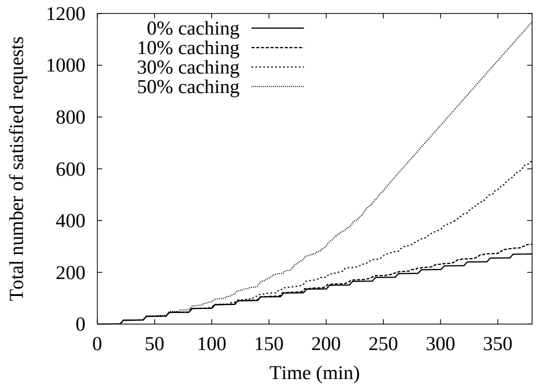
(a) Clients arrival pattern: constant rate arrivals



(b) Average waiting time

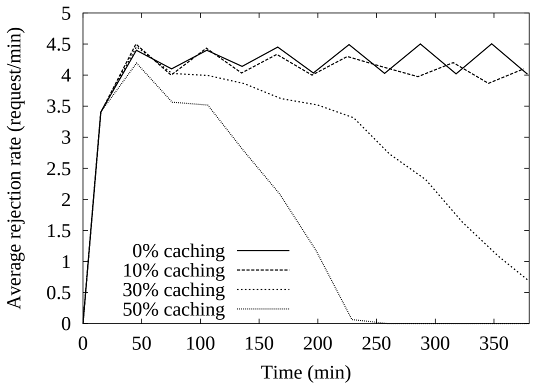


(c) Average service rate

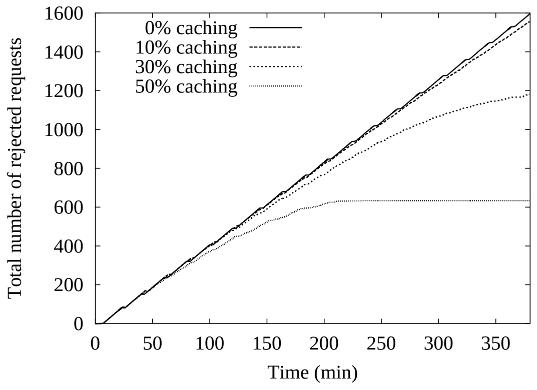


(d) Total number of satisfi ed requests

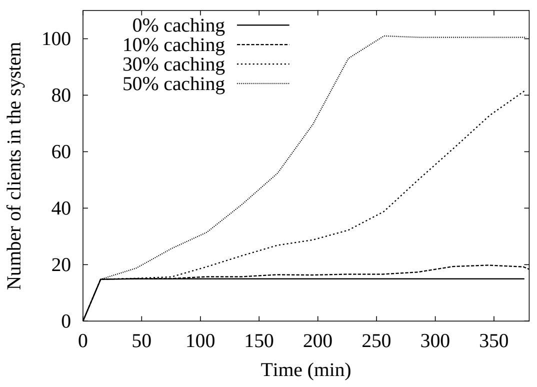
Figure 12: Performance of the P2P architecture under constant rate arrivals.



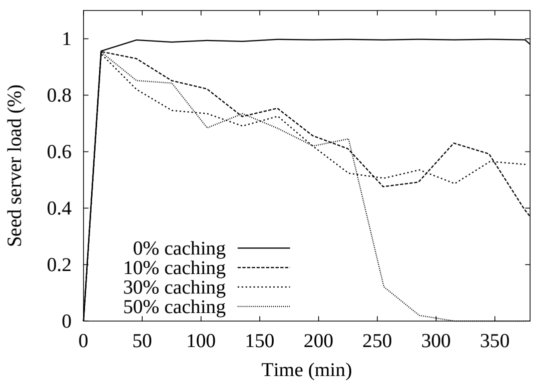
(a) Average rejection rate



(b) Total number of rejected requests



(c) Number of clients concurrently being served



(d) Load on the seeding server

Figure 13: Performance of the P2P architecture under constant rate arrivals (cont'd).

6.1.4 Results for Constant Rate Arrivals

Figure 12 and Figure13 show the behavior of the P2P architecture when the constant rate arrival pattern shown in Figure 12.a is applied to the system.

Figure 12.c and Figure 12.d show how the system's capacity evolves over the time. The average service rate, shown in Figure 12.c, increases with the time, because as the time passes more peers join the system and contribute some of their resources to serve other requesting peers. The capacity is rapidly amplified, especially with high caching percentage. For instance, with 50% caching, the system is able to satisfy all the requests submitted at 5 requests/minute after about 250 minutes (about 4.2 hours) from the starting point. We can use Figure 12.c to answer the following two questions. Given a target client arrival rate, what should be the appropriate caching percentage? How long will it take for the system to reach the steady state? To illustrate, suppose that the target service rate is 2 requests/minute. Then, 30% caching will be sufficient and the steady state will be achieved within less than 5 hours. The average waiting time, shown in Figure 12.b, is decreasing over the time, even though the system has more concurrent clients, as shown in Figure 13.d. This is due to the rapid amplification of the capacity.

Figures 13.a and 13.b complement Figures 12.c and 12.d by showing that the average rejection rate and the total number of denied requests are decreasing over the time. Finally, Figures 13.c and 13.d verify the diminishing role of the seeding server. Although the number of *simultaneous* clients increases until it reaches the maximum (limited by the arrival rate), the proportion of these clients that are served by the seeding server decreases over the time, especially with high caching percentages. For instance, with 50% caching and after about 5 hours, we have 100 concurrent clients, i.e., 6.7 times the original capacity, and none of them is served by the seeding server. Reducing the load on the seeding server is an important feature of the P2P streaming architecture, because it means that the seeding servers need not to be powerful machines with high network connectivity. Besides being moderate machines, the seeding servers are used only for a short period of time. Therefore, the cost of deploying and running these seeding servers (in case of a commercial service) is greatly reduced. Our cost-profit analysis presented in Section 5 confirms this fact.

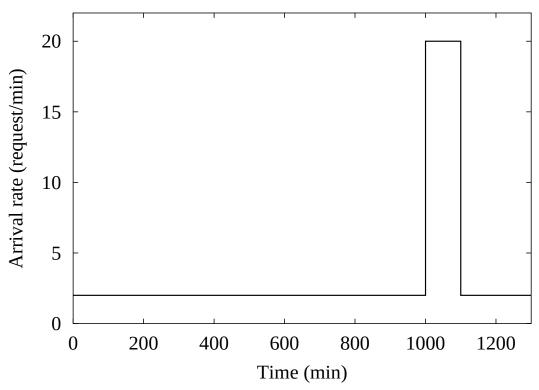
6.1.5 Results for Flash Crowd Arrivals

Flash crowd arrivals are characterized by a surge increase in the clients arrival rates. These kind of arrival patterns arise in cases such as releasing a popular movie or a recording of a publically interesting event. To simulate the flash crowd arrivals, we initially subject the system to a small request rate of 2 requests/minute for some period of time (*warm up* period), and then suddenly increase the arrival rate 10 fold to 20 requests/minutes for a limited time (100 minutes). The arrival pattern is shown in Figure 14.a.

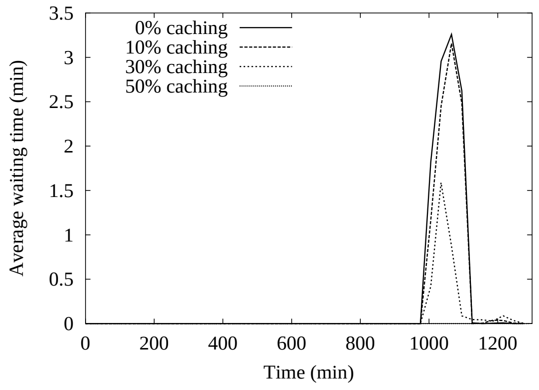
The results shown in Figure 14 and Figure 15 demonstrate an appealing characteristic of the P2P architecture, namely the ability to handle flash crowd arrivals. For 50% caching, the average service rate in the system, shown in Figure 14.c, reaches as high as the clients arrival rate (i.e., 20 requests/min) during the crowd period. Therefore, as shown in Figure 15.a, the system does not turn away any customers, when the caching percentage is 50%. Moreover, all customers are served without having to wait, as shown in Figure 14.b.

During the crowd period and with 50% caching, Figure 15.c indicates that there are as many as 400 concurrent clients in the system. This is an increase of 26.7 times in the original capacity. Even with that many clients, Figure 15.d shows that none of the clients is being served by the seeding server, which confirms that the seeding server's role is still just seeding the media file into the system.

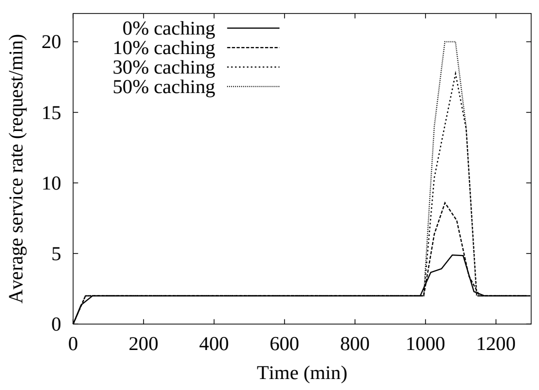
Finally, we notice that for caching percentages lower than 50%, the system needs a longer *warm up* period to cope with the flash crowd without the help of the seeding server.



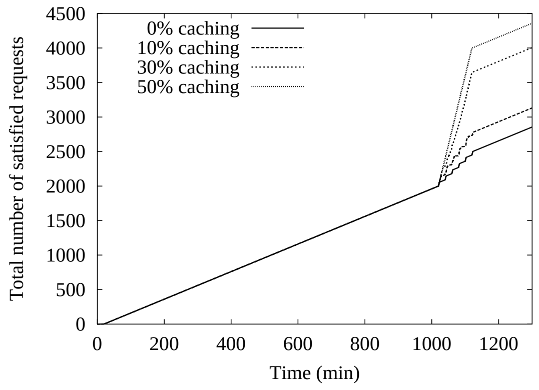
(a) Clients arrival pattern: flash crowd arrivals



(b) Average waiting time

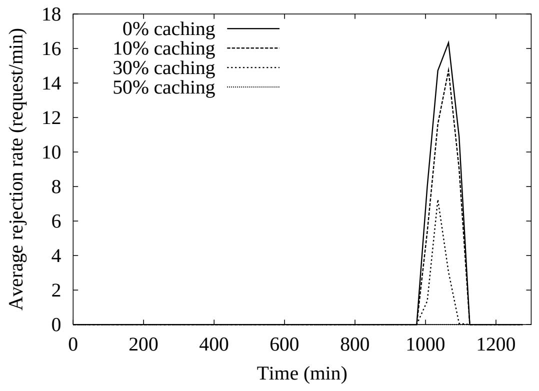


(c) Average service rate

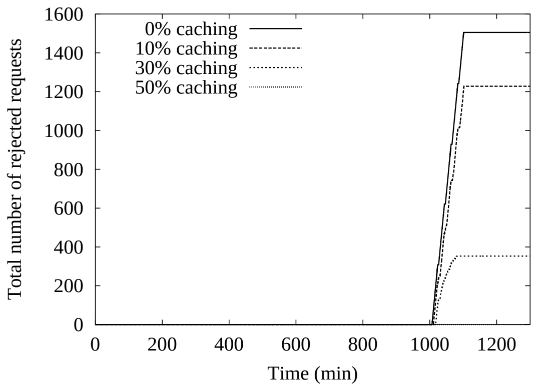


(d) Total number of satisfi ed requests

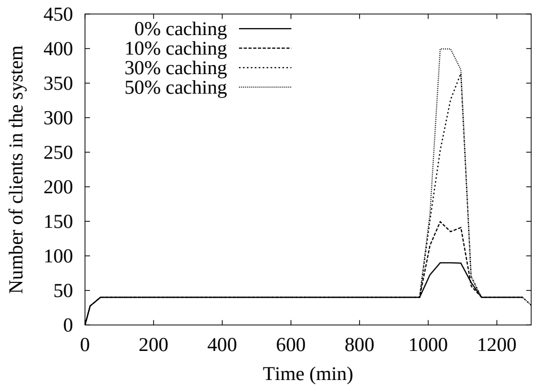
Figure 14: Performance of the P2P architecture under flash crowd arrivals.



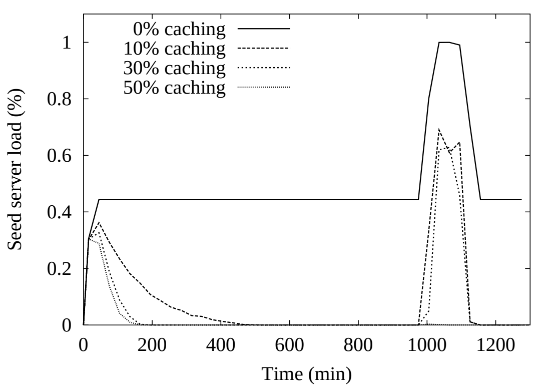
(a) Average rejection rate



(b) Total number of rejected requests

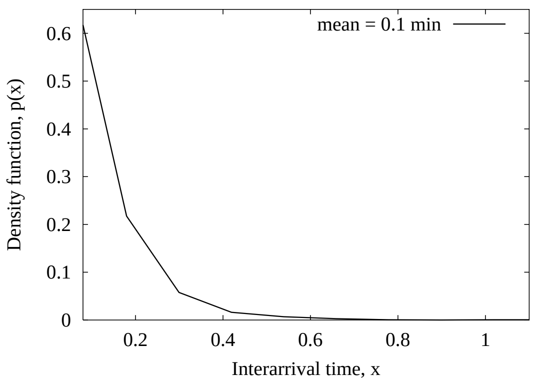


(c) Number of clients concurrently being served

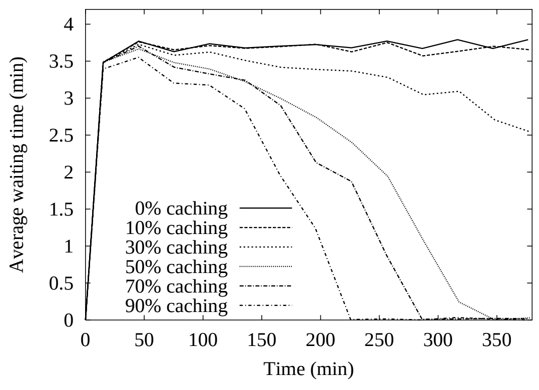


(d) Load on the seeding server

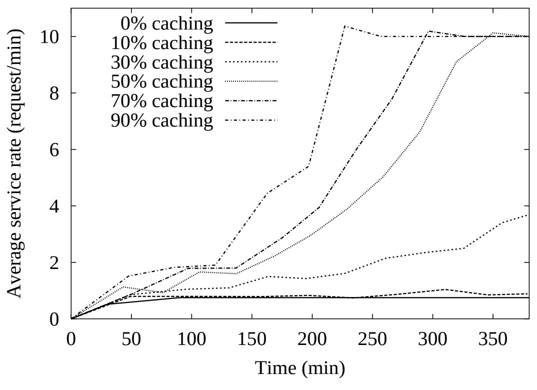
Figure 15: Performance of the P2P architecture under flash crowd load (cont'd).



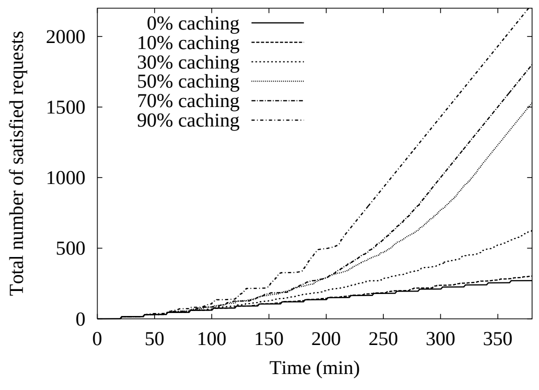
(a) Density function of the inter-arrival time. (Exponential with mean 0.1)



(b) Average waiting time



(c) Average service rate



(d) Total number of satisfi ed requests

Figure 16: Performance of the P2P architecture under Poisson Arrivals, mean inter-arrival time = 0.1 minute.

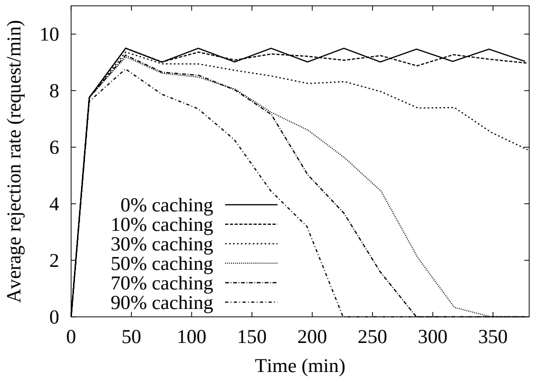
6.1.6 Results for Poisson Arrivals

We subject the system to Poisson arrivals with different mean arrival rates. The results for mean arrival rate of 10 requests/minute (i.e., mean inter-arrival time of 0.1 minute) are shown in Figure 16 and Figure 17. Notice that, Figure 16.a shows the density functions of the inter-arrival time distribution (exponential distribution). The results are similar to the case of constant rate arrivals, except there are more fluctuations due to the probabilistic nature of the Poisson arrivals. The results indicate the ability of the P2P architecture to handle statistically multiplexed client arrival patterns.

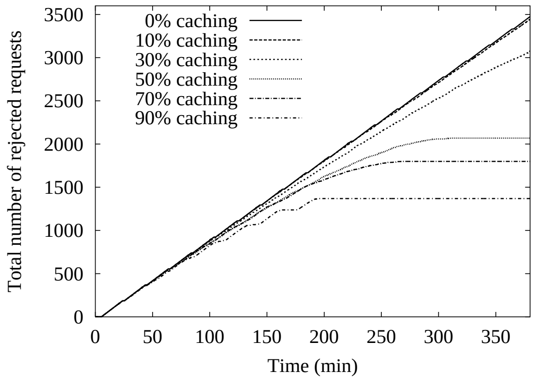
6.2 Evaluation of the Dispersion Algorithm

Since we are not aware of any existing dispersion algorithms that can be used in our model, we compare our cluster-based dispersion algorithms against a random dispersion algorithm. Both the `IndexDisperse` and the `OverlayDisperse` dispersion algorithms try to keep copies of the requested files as close as possible to the clients, i.e., within the same network domain. In the following comparison, we evaluate the overlay version, which does not need a global index. Thus, we are evaluating the *weaker* version of our algorithm.

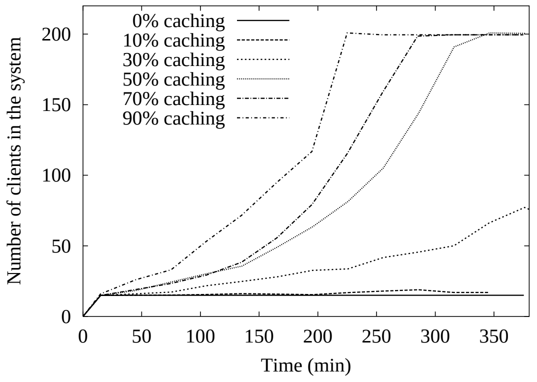
We evaluate the efficiency of the dispersion algorithm by measuring the average number of network



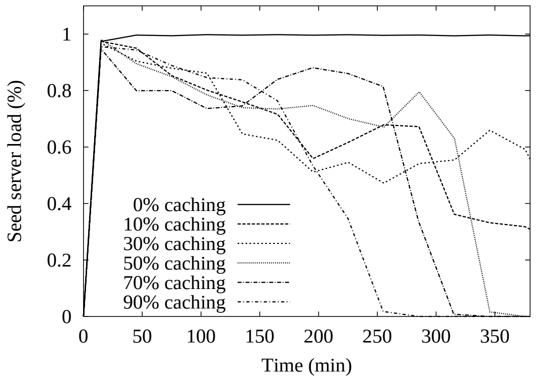
(a) Average rejection rate



(b) Total number of rejected requests

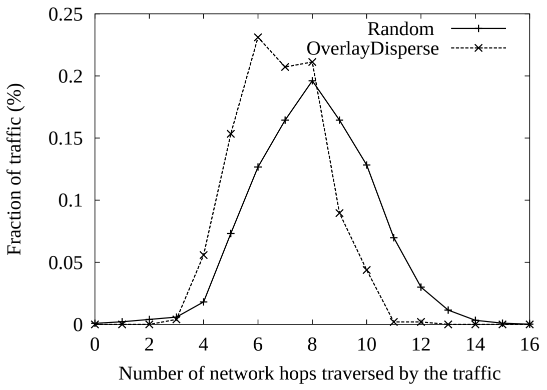


(c) Number of clients concurrently being served

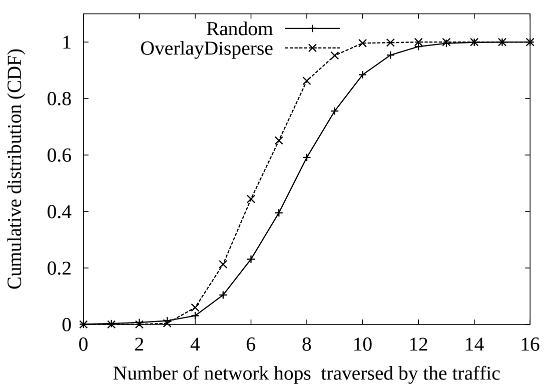


(d) Load on the seeding server

Figure 17: Performance of the P2P architecture under Poisson Arrivals, mean inter-arrival time = 0.1 minute (cont'd).



(a) Probability mass function (pmf) of the number of network hops



(b) Cumulative distribution function (CDF) of the number of network hops

Figure 18: Comparison between the random and the `OverlayDisperse` dispersion algorithms, 5% caching.

hops traversed by the requested stream. Smaller number of network hops indicates savings in the backbone bandwidth and less susceptibility to congestion, since traffic passes through fewer routers.

6.2.1 Topology and Simulation Parameters

In this set of experiments, most of the parameters are the same as in the previous set of experiments, except that the topology is larger. The topology has 100 transit domains, 400 stub domains, 2,400 routers, and a total of 12,021 hosts. This topology is chosen to distribute the peers over a wider range, and hence stresses the dispersion algorithms more than the previous topology.

We vary the caching percentages from 5% to 90%. Low caching percentages, e.g., 5% and 10%, stress the dispersion algorithm more than the higher caching percentages. With low caching percentages, a peer stores few segments. Therefore, it is important for the dispersion algorithm to carefully choose these few segments. In contrast, with high caching percentages, a peer stores most of the segments, leaving little work for the dispersion algorithm.

The clients arrive to the system according to a constant rate arrival pattern with a rate of 1 request/minute.

6.2.2 Simulation Scenario

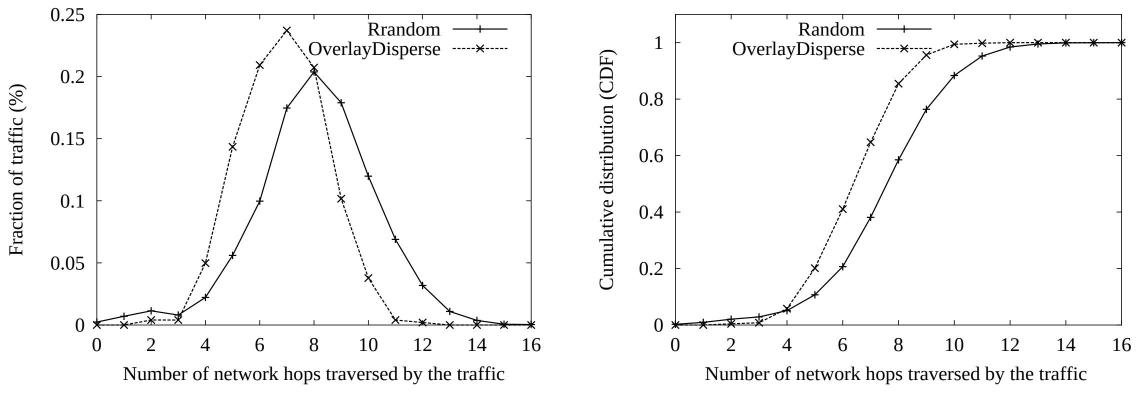
The simulation scenario is similar to the scenario in the previous set of experiments with one difference in the last step of the `P2PStream` protocol (the caching step). For each caching percentage, we run the experiment twice. In the first run, we use a random dispersion algorithm, in which a peer *randomly* selects a specific number of segments (determined by the caching percentage) and store them locally. In the second run, we use the `OverlayDisperse` algorithm, which caches the same number of segments but selects them carefully. The `OverlayDisperse` algorithm gives more preference to segments obtained from far away sources (in terms of number of network hops) than those obtained from near sources (see Section 4.2.1 for details).

Each experiment lasts for 500 minutes of simulation time. For every streaming packet transmitted during the simulation, we measure the number of network hops that packet traverses. At the end of each experiment, we compute the distribution of the number of network hops traversed by all packets of the streaming traffic. We plot both the probability mass function (pmf) and the cumulative distribution function (CDF). The results are summarized in the following subsections.

6.2.3 Results for 5% Caching

Figure 18.a shows the pmf of the number of network hops for both the random and the `OverlayDisperse` dispersion algorithms. The pmf curve of the `OverlayDisperse` algorithm is shifted to the left of the random algorithm. This indicates that the traffic crosses fewer number of hops using the `OverlayDisperse` algorithm than using the random algorithm. The arithmetic mean of the number of network hops for the random algorithm is 8.0520, while it is 6.8187 for the `OverlayDisperse` algorithm. The saving is about 15.3% of the total bandwidth needed in the backbone. Given that a good streaming service requires a huge bandwidth, our dispersion algorithm achieves considerable savings.

The cumulative distribution, Figure 18.b, shows that about 44% of the traffic crosses six or less hops using our algorithm, whereas this value is only 23% for the random algorithm. A reasonable ISP network would have an average network diameter in the vicinity of six hops. This means that our dispersion algorithm keeps about 44% of the traffic within the same domain (cluster), which is often a desirable property for both the clients and the network.



(a) Probability mass function (pmf) of the number of network hops

(b) Cumulative distribution function (CDF) of the number of network hops

Figure 19: Comparison between the random and the `OverlayDisperse` dispersion algorithms, 10% caching.

6.2.4 Results for other Caching Percentages

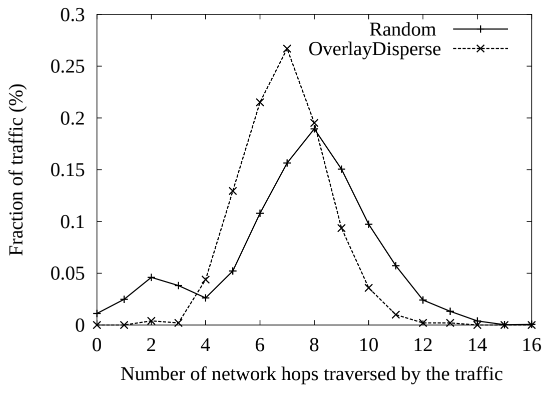
Similar results were obtained for other caching percentages. To conserve space, we present only the results for 10% caching (Figure 19) and for 30% caching (Figure 20). As shown in Figure 19 and Figure 20, the `OverlayDisperse` algorithm still outperforms the random dispersion algorithm. The main observation is that the difference between the two algorithm is shrinking as the caching percentage increases. This is expected, since peers cache more segments as the caching percentage increases and the room for enhancements by the dispersion algorithm is decreased.

7 Related Work

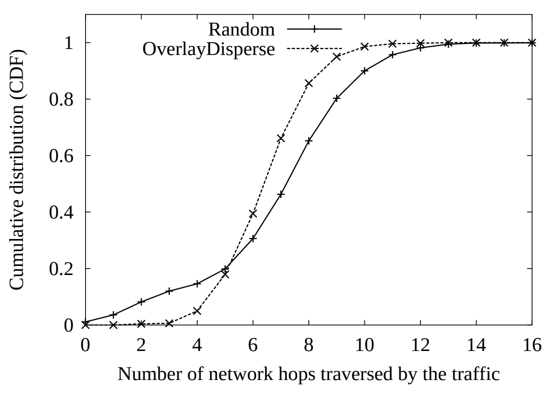
Significant research effort has addressed the problem of efficiently streaming multimedia, both live and on demand, over the best-effort Internet. Directly related to our work are systems like *SpreadIt* [5] for streaming live media and *CoopNet* [14], [13] for both live and on-demand streaming. Both systems build distribution trees using application-layer multicast and, like ours, they rely on cooperating peers. Multicast (network- or application-layer) is the basis for several other media delivery systems [6] [7]. Our work is different from these systems, since we do not use multicast in any form and our system is more appropriate for on-demand media service.

In the client/server world, proxies and caches are deployed at strategic locations in the Internet to reduce and balance load on servers and to achieve a better service. Content Delivery Network (CDN) companies such Akamai [15] and Digital Island [16] follow similar approaches to provide media streaming and other services. Our approach does not require any powerful proxies or caches. Rather, it uses peers' extra resources as numerous tiny caches. These tiny caches do not require large investment and collectively enlarge the capacity of the system in a way that potentially outperforms any powerful centralized caches.

The distributed video streaming framework [12] is also relevant to our work. The framework allows for multiple senders to feed a single receiver. The receiver uses a rate allocation algorithm to specify the sending rate for each sender to minimize the total packet loss. This specification is based on estimating the loss rate and the available bandwidth between the receiver and each of the senders. The authors assume that senders are capable of providing the rates computed by the rate allocation algorithm. In our case, the providing peers



(a) Probability mass function (pmf) of the number of network hops



(b) Cumulative distribution function (CDF) of the number of network hops

Figure 20: Comparison between the random and the `OverlayDisperse` dispersion algorithms, 30% caching.

decide on the rates at which they are willing to provide. However, the rate allocation algorithm can be used to enhance our model in a special case, namely, when a requesting peer receives offers from multiple providing peers with an aggregate rate exceeding the one required for streaming.

Recently, the peer to peer paradigm has attracted the attention of numerous researchers. Two main categories of research can be identified: research on protocols and algorithms (mainly on searching), and research on building P2P systems. Location and routing protocols such as CAN [21], Chord [26], Pastry [22], and Tapestry [29] guarantee locating the requested object within a logarithmic number of steps, if the object exists in the system. However, they lack the flexibility of supporting keyword queries and in many cases they do not consider network locality. Other searching techniques do not provide such guarantees but they support flexible queries [28]. The overlay architecture discussed in Section 4.2 can use any of these protocols. On the systems side, Gnutella [18] is the largest currently running file-sharing system. Freenet is another file-sharing system focusing on the *anonymity* of both the producer and consumer of the files [17]. Examples of large-scale storage systems built on top of P2P architectures are presented in [4], [11], and [23]. Our proposed system adds one more to the list but with a new service, namely, media streaming.

The cost comparisons presented in [1] and [3] show that a distributed architecture for video on-demand services has a lower cost than a centralized one. Our cost-profit analysis shows the potential of a P2P architecture for a higher cost effectiveness compared to the distributed approach, because the P2P architecture does not require powerful caches or proxies. It needs, however, limited-capacity seeding servers for a short time, which cost much less than powerful caches.

8 Conclusions and Future Work

We presented a P2P media streaming model that can serve many clients in a cost effective manner. We presented the details of the model and showed how it can be deployed over the current Internet. Specifically, we presented two architectures to realize the model, index-based and overlay. We presented the necessary protocols and algorithms for both architectures, namely, the dispersion and searching algorithms. Through large-scale simulation, we showed that our model can handle several types of client arrival patterns, including suddenly increased arrivals, i.e., flash crowds. Our simulation also showed that the proposed cluster-based dispersion algorithm reduces the load on the underlying network and keeps a large portion of the traffic within the same network domain.

Our approximate cost-profit analysis showed the economic potential of a large-scale media streaming service built on top of a P2P infrastructure. We showed that using the inducement factor, the provider can manage the level of cooperation offered by peers. Specifically, our analysis can be used in two ways. First, if we have sufficiently large number of customers in a given period, the *optimal* value of the inducement factor α_{opt} that maximizes the provider's revenue can be computed. Second, if there is a limit on the expected number of customers, the *appropriate* value of α can be chosen to ensure the sufficient capacity within the target period. We compared the profits that can be obtained in the P2P architecture and the conventional client/server architecture. Our comparison showed that even with large inducement factors (i.e., peers get a larger portion of the per-customer profit than the provider), the P2P architecture still achieves a higher profit with a reasonable number of customers. The comparison also demonstrated the cost-effectiveness of the P2P architecture, since it requires a small initial investment in the infrastructure.

We are currently embarking on implementing a prototype of the P2P media streaming system. The objective is to better assess to the proposed model and to demonstrate its applicability for a wide deployment. Addressing the security and robustness issues of the model are parts of our future work. The main extension of our cost-profit analysis is to consider the case of multiple movies. Specifically, the relationship between the inducement factor and the relative popularities of the movies needs to be studied and included in the analysis. Finally, obtaining a more realistic data on the reactions of the peers to the offered inducement factor will make the analysis more realistic. This can be done by obtaining a survey from a random sample of customers.

Acknowledgments

The authors would like to thank Leszek Lilien, Ahsan Habib, and Florian Baumgartner for their valuable comments.

References

[1] S. Barnett and G. Anido. A cost comparison of distributed and centralized approaches to video-on-demand. *IEEE Journal on Selected Areas in Communications*, 14(6):pp. 1173–1183, August 1996.

[2] K. Calvert, M. Doar, and E Zegura. Modeling internet topology. In *IEEE Communications Magazine*, pages 35:160–163, 1997.

[3] S. Chan and F. Tobagi. Distributed servers architecture for networked video services. *IEEE Transactions on Networking*, 9(2):125–136, April 2001.

[4] F. Dabek, M. Kaashoek, D. Karger, D. Morris, I. Stoica, and H. Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. In *Proc. of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII*, pages 71–76, Elmau/Oberbayern, Germany, May 2001.

[5] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over peer-to-peer network. Technical report, Stanford University, 2001.

[6] A. Dutta and H. Schulzrinne. A streaming architecture for next generation internet. In *Proc. of ICC'01*, Helsinki, Finland, June 2001.

[7] L. Gao and D. Towsley. Threshold-based multicast for continuous media delivery. *IEEE Transactions on Multimedia*, 3(4):pp. 405–414, December 2001.

[8] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives for sharing in peer-to-peer networks. In *Proc. of ACM Conference on Electronic Commerce (EC'01)*, pages 14–17, Tampa, FL, USA, October 2001.

[9] S. Gribble, A. Havely, Z. Ives, M. Rodrig, and Suciu D. What can databases do for peer-to-peer? In *Proc. of WebDB Workshop on Databases and the Web*, 2001.

[10] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *Proc. of ACM SIGCOMM*, Stockholm, Sweden, August 2000.

[11] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proc. of Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, pages 190–201, Boston, MA, November 2000.

[12] T. Nguyen and A. Zakhor. Distributed video streaming over internet. In *Proc. of Multimedia Computing and Networking (MMCN02)*, San Jose, CA, USA, January 2002.

[13] V. Padmanabhan and K. Sripanidkulchai. The case for cooperative networking. In *Proc. of The 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, USA, March 2002.

[14] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proc. of NOSSDAV'02*, Miami Beach ,FL, USA, May 2002.

[15] Akamai Home Page. http://www.akamai.com.

[16] Digital Island Home Page. http://www.digitalisland.com.

[17] Freenet Home Page. http://www.freenet.sourceforge.com.

[18] Gnutella Home Page. http://www.gnutella.com.

[19] Napster Home Page. http://www.napster.com.

[20] PIMSS Home Page. http://www.cs.purdue.edu/homes/mhefeeda.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, San Diago, CA, USA, August 2001.

[22] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, November 2001.

[23] A. Rowstron and P. Druschel. Storage management in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of the 18th ACM Symposium on Operating Systems Principles*, October 2001.

[24] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking (MMCN02)*, San Jose, CA, USA, January 2002.

[25] The Network Simulator. http://www.isi.edu/nsnam/ns/.

[26] I. Soitca, R. Morris, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, San Diago, CA, USA, August 2001.

[27] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On peer-to-peer media streaming. In *Proc. of IEEE ICDCS*, Vienna, Austria, July 2002.

[28] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proc. of ICDCS'02*, Vienna, Austria, July 2002.

[29] B. Zaho, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.