CERIAS Tech Report 2004-102
A Cost-Effective Architecture for On-demand Media Streaming
by M Hefeeda, B Bhargava, D Yau
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Computer Networks 44 (2004) 353–382
COMPUTER NETWORKS
www.elsevier.com/locate/comnet

A hybrid architecture for cost-effective on-demand media streaming

Mohamed M. Hefeeda *, Bharat K. Bhargava, David K.Y. Yau

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA

Received 11 November 2002; received in revised form 17 June 2003; accepted 1 October 2003

Responsible Editor: N. Georganas

Abstract

We propose a new architecture for on-demand media streaming centered around the peer-to-peer (P2P) paradigm. The key idea of the architecture is that peers *share* some of their resources with the system. As peers contribute resources to the system, the overall system capacity increases and more clients can be served. The proposed architecture employs several novel techniques to: (1) use the *often-underutilized* peers' resources, which makes the proposed architecture both deployable and cost-effective, (2) aggregate contributions from multiple peers to serve a requesting peer so that supplying peers are not overloaded, (3) make a good use of peer heterogeneity by assigning relatively more work to the powerful peers, and (4) organize peers in a *network-aware* fashion, such that nearby peers are grouped into a logical entity called a *cluster*. The network-aware peer organization is validated by statistics collected and analyzed from real Internet data. The main benefit of the network-aware peer organization is that it allows to develop efficient searching (to locate nearby suppliers) and dispersion (to disseminate new files into the system) algorithms. We present network-aware searching and dispersion algorithms that result in: (i) fast dissemination of new media files, (ii) reduction of the load on the underlying network, and (iii) better streaming service.

We demonstrate the potential of the proposed architecture for a large-scale on-demand media streaming service through an extensive simulation study on large, Internet-like, topologies. Starting with a limited streaming capacity (hence, low cost), the simulation shows that the capacity rapidly increases and many clients can be served. This occurs for all studied arrival patterns, including constant rate arrivals, flash crowd arrivals, and Poisson arrivals. Furthermore, the simulation shows that a reasonable client-side initial buffering of 10–20 s is sufficient to ensure full quality playback even in the presence of peer failures.

Keywords: Media streaming; Peer-to-peer systems; Multimedia systems; Peer clustering; Dispersion algorithms

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

* Corresponding author.
E-mail addresses: mhefeeda@cs.purdue.edu (M.M. Hefeeda), bb@cs.purdue.edu (B.K. Bhargava), yau@cs.purdue.edu (D.K.Y. Yau).

1389-1286/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2003.10.002

and cost-effective media streaming architecture over the current Internet a challenge. In this paper, we target on-demand streaming environments such as the one shown in Fig. 1. Examples of this environment include a university distance learning service and an enterprise streaming service. In this kind of environment, the media contents are streamed to many clients distributed over several campuses or branches in the Internet.

Before we proceed, we clarify the differences between a P2P file-sharing system and a P2P media streaming system [41]. In file-sharing systems such as Gnutella [26] and Kazaa [27], a client first downloads the *entire* file before using it. The shared files are typically small (a few Mbytes) and 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 the file. 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 can tolerate inter-packet delays. In media streaming systems, a client *overlaps* downloading with the consumption of the file. 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 a 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 playback time is useless.

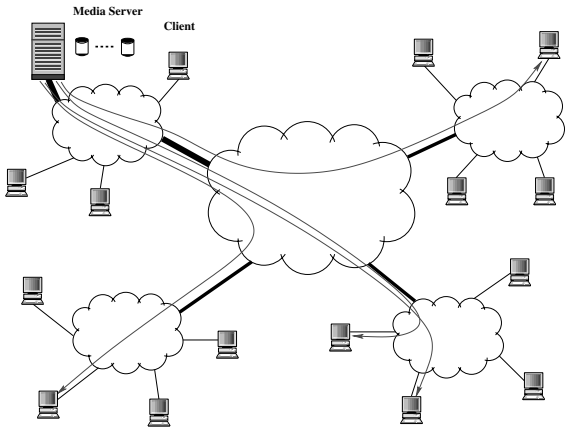


Fig. 1. The target environment for the proposed streaming architecture. Examples include a university distance learning service and a corporate streaming service.

There are several approaches that can be used to stream media to the clients in the target environment. We start by briefly describing the current approaches in the literature. The objective is to highlight the key ideas and limitations of each approach and to position our proposed approach in the appropriate context within the global picture. We can roughly categorize the current approaches into two categories: unicast-based and multicast-based.

1.1. Unicast-based approaches

In these approaches a unicast stream is established for every client. Roughly, there are three approaches that use unicast for on-demand streaming: *centralized*, *proxy*, and *content distribution networks* (CDN).

Centralized. The straightforward centralized approach (Fig. 1) is to deploy a powerful server with a high-bandwidth connection to the Internet. This approach is easy to deploy and manage. However, the scalability and reliability concerns are obvious. The reliability concern arises from the fact that only one entity is feeding all clients; i.e., there is 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 large-scale potential users, since adding more users requires adding a commensurate amount of resources to the supplying server. There are two other critical, but less obvious, disadvantages of the centralized approach: *high cost* and *load on the backbone network*. To appreciate the cost issue, consider, for instance, a streaming server connected to the Internet through a T3 link ($\sim$45 Mb/s), which is a decent and expensive link. This server would be able to support up to 45 concurrent users requesting media files recorded at 1 Mb/s, assuming that the CPU and I/O support that. Since all clients have to go to the server for all requests, much traffic will have to travel through the wide-area network. This adds to the cost of streaming and increases the load on the backbone network. In addition, when the traffic travels

through many network hops, it will be susceptible to higher delay variations and packet losses due to possible congestion in the Internet.

Proxy. In the proxy approach [13,17,36,39], proxy servers are deployed near the client domains (Fig. 2). Since movies are large in size, the proxy may be able to cache a few movies in their entirety. A number of caching techniques have been proposed to enable the proxy to cache a fraction of each movie, and therefore more movies can be cached. In prefix caching [36], the proxy stores the first few frames of the movie allowing for short startup delays. In staging caching [39], the proxy stores the bursty portions of the frames and leaves the smoother parts on the central server. This alleviates the stringent bandwidth requirements on the WAN links. A non-contiguous selection of *intermediate* frames can also be cached [17], which facilitates control functions such as fast forward and rewind. The proxy approach and its variations save WAN bandwidth and is expected to yield short startup delay and small jitter. On the negative side, this approach requires deploying and managing proxies at many locations. While deploying proxies increases the overall system capacity, it multiplies the cost. The capacity is still limited by the aggregate resources of the proxies. This shifts the bottleneck from one central point to a "few" distributed points, but does not eliminate it.

Content distribution network. The third unicast approach employs a *third-party* for delivering the contents to the clients. This third party is known as a content delivery network (CDN). Content delivery networks, such as Akamai and Digital Island, deploy thousands of servers at the *edge* of the Internet (see Fig. 3). Akamai, for instance, deploys more than 10,000 servers [1]. These servers (also called caches) are installed at many POPs (point of presence) of major ISPs such as AT&T and Sprint. The idea is to keep the contents close to the clients, and hence traffic traverses fewer network hops. This reduces the load on the backbone network and results in a better service in terms of shorter delay and smaller loss rate. The CDN caches the contents at many servers and redirects a client to the most suitable server. Proprietary protocols are used to distribute contents over servers, monitor the current traffic situation over the Internet, and direct clients to servers. Cost-effectiveness is a major concern in this approach, especially for distributing large files such as movies: the CDN operator charges the content provider for every megabyte served. This delivery cost might be acceptable for relatively small files such as web pages with some images. However, it would render a costly streaming service for the targeted environment because media files are typically large.

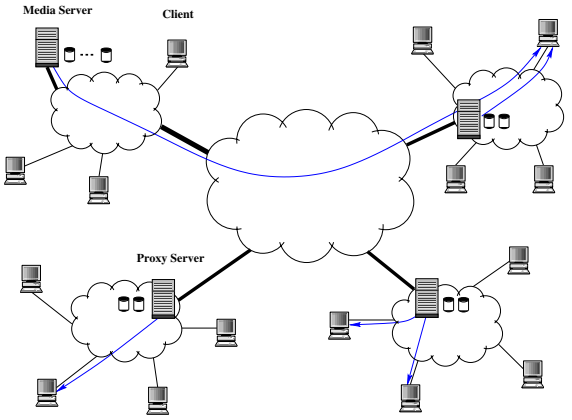


Fig. 2. Proxy-based architecture for distributing media.

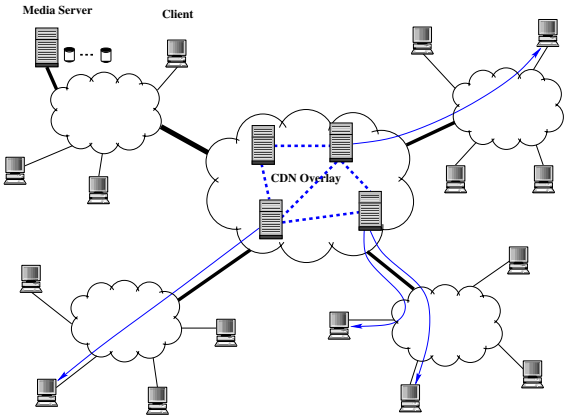


Fig. 3. The CDN approach for distributing media.

1.2. Multicast-based approaches

The multicast approaches achieve better resource utilization by serving multiple clients using the same stream. The basic idea is to establish a multicast session to which clients subscribe. This is done by creating multicast distribution trees. Multicast approaches are more natural to live streaming in which clients are synchronized: all clients receive the same portions of the stream at the same time. To cope with the asynchronous nature of the on-demand service, several techniques have been proposed. One of the key ideas in adapting multicast to on-demand service is *patching* and its variations [12,35]. A good comparison is given in [16]. In patching (also known as tapping), a new client arriving within a threshold is allowed to join an on-going multicast session. In addition, the client establishes a unicast connection with the server to "patch" or get the missed part of the file. The two streams run at the full play rate. The patch stream terminates when the client gets the missed part. Patching techniques may require the client to tune in to multiple streams during the patching period. This means that the client has to have an inbound bandwidth of at least double the streaming rate. This is quite a stringent requirement for the limited-capacity peers in the target environment.

Multicast distribution trees are either created at the network level (Fig. 4) or at the application level (Fig. 5). The network-level multicast establishes a tree over the internal routers with the clients as the leaves of the tree. While network-level multicast is efficient, it is not widely deployed. For the target environment (e.g., distance learning), some of the intermediate network domains may not support multicast. Therefore, currently, network-level multicast is not a feasible solution for delivering contents to all the clients.

Application-level multicast techniques, such as NICE [2], Narada [7], Zigzag [38], among others, construct the distribution trees over the end systems. The algorithms used to construct the distribution tree (or mesh in the case of Narada) differ from one technique to another. Building the tree over the end systems achieves deployability in the current Internet. However, it introduces another problem: it may overload some end systems beyond their capacities. An end system in the tree may become a parent of several other end systems. For example, in Fig. 5, peer P1 is the parent of peers P2 and P3. Hence, P1 should be able to provide the stream to both of them. This assumes that P1 can (and is willing to) support multiple folds of the streaming rate. In the target environment, nodes typically have limited capacity, especially of the upstream bandwidth. In many cases, nodes cannot even provide the full stream rate to another node.

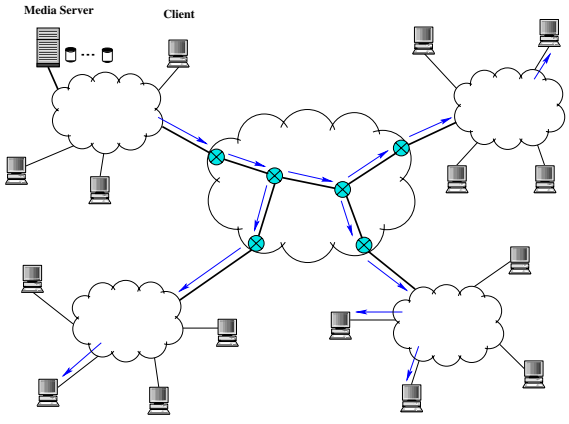


Fig. 4. Network-level multicast for distributing media.

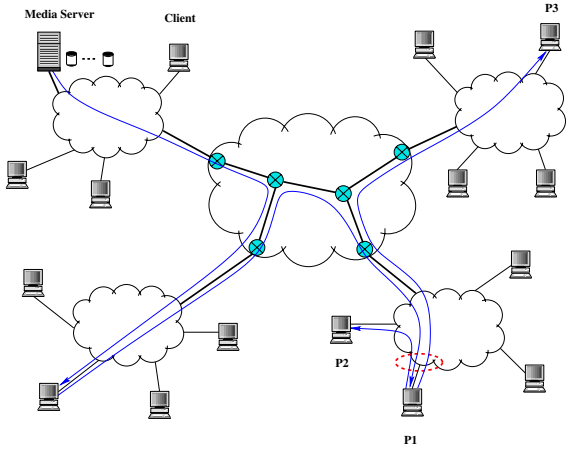


Fig. 5. Application-level multicast for distributing media.

1.3. The proposed architecture

We propose a new peer-to-peer (P2P) media distribution architecture that can support a large number of clients at a low overall system cost. The key idea of the architecture is that end systems (called peers hereafter) *share* some of their resources with the system. As peers contribute resources to the system, the overall system capacity increases and more clients can be served. As shown in Fig. 6, most of the requesting peers will be served using resources contributed by other peers. The proposed architecture employs several novel techniques to avoid the limitations of the current approaches. Specifically, it has techniques to:

(i) Use the *often-underutilized* peer resources, which makes the proposed architecture both deployable and cost-effective. It is deployable because it does not need any support from the underlying network: all work is done at the peers. Since the architecture neither needs new hardware to be deployed nor requires powerful servers, it is highly cost-effective.
(ii) Aggregate contributions from multiple peers to serve a requesting peer. This indicates that a single supplying peer may only serve a fraction of the full request. Moreover, the requesting peer is not required to have an extra inbound bandwidth to get full-quality streaming.

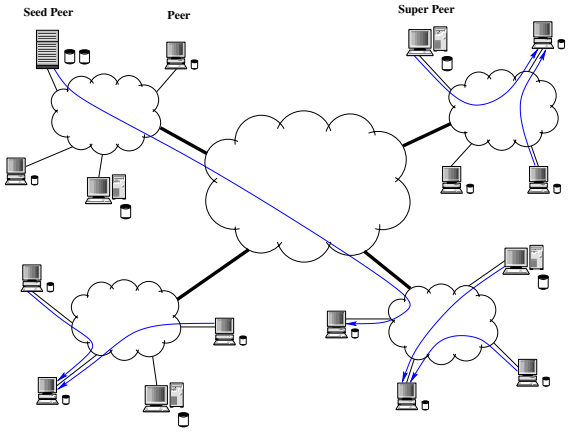


Fig. 6. The proposed hybrid architecture for distributing media.

(iii) Organize peers in a *network-aware* fashion, in which nearby peers are grouped into a logical entity called a cluster. This organization of peers is validated by statistics collected and analyzed from real Internet data. The main benefit of the network-aware peer organization is that it allows for developing efficient searching (to locate nearby suppliers) and dispersion (to disseminate new files into the system) algorithms. Network-aware searching and dispersion result in two desirable effects: (1) reduction of the load on the underlying network, since the traffic traverses a fewer number of hops, and (2) better streaming service because the delay is shorter and less variable.
(iv) Make good use of peer heterogeneity [34]. The architecture assigns relatively more work to the powerful peers. Specifically, powerful peers help in the searching and the dispersion algorithms. This special assignment makes the proposed architecture not purely P2P. Therefore, in the rest of the paper we will call it the *hybrid architecture*.

The P2P architecture has the potential to provide the desired large-scale media distribution service, especially for the environments considered in this paper (Fig. 1). These environments assume that peers can be asked/configured to share resources. For example, a university distance learning service may configure peers at remote campuses to share storage and bandwidth. This is not a concern, since these peers are owned by the university. The architecture may also be extended to provide a commercial service. However, in this case peer *rationality* or self-interest should be considered, since peers may not voluntarily contribute resources to the system. This requires developing economic *incentive* mechanisms to properly motivate peers. One such mechanism is the *revenue sharing* proposed in [11]. In the revenue-sharing mechanism, the provider *shares* part of the revenue from serving clients with the peers who helped in serving those clients.

The rest of this paper is organized as follows. Section 2 provides an overview of the proposed architecture. The network-aware organization of

peers is detailed in Section 3. The searching and the dispersion algorithms are presented in Section 4. The simulation study is presented in Section 5. Section 6 summarizes related research effort. Section 7 concludes and proposes future extensions for this research.

2. The hybrid architecture for media streaming

This section provides an overview of the proposed architecture. It first identifies all entities in the system. Then, it explains how the streaming sessions are established and the effect of peer failures on the quality of playback.

2.1. System entities and their roles

In the following, we define the entities participating in our architecture, their roles, and how they interact with each other.

- *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 the streaming center. Let $\mathbb{P} = \{P_1, P_2, \ldots, P_{\mathcal{N}}\}$ be the set of all peers in the system. Peers in P2P systems have been shown to be quite heterogeneous [34]. We model this heterogeneity as follows. Every peer P_i, $1 \leqslant i \leqslant \mathcal{N}$, specifies three parameters: (1) R_i (in Kb/s), the maximum bandwidth 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 media files; and (3) C_i, the maximum number of concurrent connections that can be opened to serve requesting peers. By using these three parameters, a peer has the ability to control its level of cooperation with other peers in the system.
- *Seed Peers*. One of the peers or a subset of them that seed new media files in the system. We choose the name *seeding* peers to indicate that their main functionality is to *initiate* the streaming service and not to serve all clients at all times. Any node (peer) in the system can introduce new media files, i.e., become a seed peer for a specific period. However, in the case of a media center distributing contents, the seed peers can be dedicated machines with reasonable capacity.
- *Super Peers*. These are nodes selected to aid in locating the requested objects and disseminating the newly published ones. Super peers maintain information about the current peers in the system and the contents stored at each of them. Each super peer is responsible for a small fraction of the peers in the system. The choice and management of super peers are detailed in Section 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. The packets should be downloaded before their scheduled display time to guarantee non-disruptive playback of the media.
- *Media files*. The set of movies currently available in the system or offered by the media center. Let $\mathbb{M} = \{M_1, M_2, \ldots, M_m\}$ be the set of all available media files in the system. Every file has a size in bytes and is recorded at a specific bit rate R (in kb/s). We assume that R represents 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. In general, one segment can be streamed to the requesting peer from multiple peers at the same time. According to our protocol (see Section 2.2), every peer will supply a different *piece* of the segment proportional to its streaming rate.

2.2. System operation

To start a streaming session, a requesting peer runs a protocol composed of three phases: availability check, streaming, and caching. In phase I, the requesting peer checks for the availability of the desired media file in the system. This is done by sending a lookup request to the super peer to which the requesting peer is attached. The super peer applies a *cluster-based* searching algorithms (Section 4) and returns a list of candidate supplying peers. The list is arranged into a two-dimen-

sional table. Each row j of the table contains information about peers that are currently caching segment s_j of the requested file. Peers in each row are divided into two groups: main suppliers and backup suppliers. The main suppliers can collectively stream the segment at the full rate. The backup suppliers are used to replace the main suppliers in case of failure or degradation.

The streaming phase (phase II) of the protocol 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 after an *initial buffering* period, as discussed in Section 2.3. For every segment s_j, the protocol concurrently connects to the main suppliers that are scheduled to provide that segment. The connections remain alive 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 $R_x \leqslant R$, 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 $\sum_{x=1}^{|\mathbb{P}^j|} |s_j|(R_x/R) \geqslant |s_j|$, all pieces of the segment will be downloaded by the end of the δ period. To illustrate, Fig. 7 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.

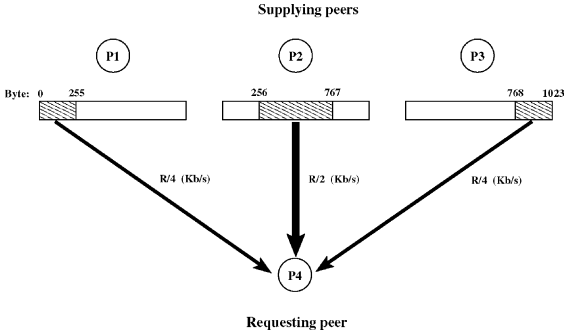


Fig. 7. Peers P_1, P_2, and P_3 serving different pieces of the same segment to peer P_4 with different rates.

Finally, in the caching phase (phase III), the peer may cache some segments in order to disseminate the file into its cluster. Which segments a peer should cache is determined by the peer's level of cooperation and the *dispersion* algorithm (Section 4).

2.3. Peer reliability and client buffering

Peers are not reliable machines: they may fail, go off-line, or stop streaming at any time. To maintain full playback quality throughout the streaming session, a quality maintenance mechanism is needed. The quality maintenance mechanism has two parts: quality degradation detection and recovery. The degradation in quality could be due to peer failures or network congestion. In either case, the aggregate rate received by the client will be smaller than the required rate R. To detect degradation, the client monitors the incoming rate from each supplier. If the incoming rate is decreasing, peer *switching* or replacement is needed: a peer from the backup suppliers is chosen to replace the degraded peer. The backup peer is notified by a control packet, which specifies the data to be sent and at what rate.

During the switching process, the aggregate received rate is less than the required playback rate and the client may experience buffer underflow. This is due to the time needed to detect quality degradation and to trigger one of the backup suppliers. We call this time the *switching time*. To hide the effect of switching, *client initial buffering* is introduced. In Section 5.2, we use simulation to study the effect of switching on the quality of playback. We also estimate the size of the initial buffering that completely offsets this effect.

3. Organization of peers

Most of the current P2P file-sharing systems do not consider network locality in their protocols. For example, in Gnutella [26], a joining peer connects itself to a few of the currently active peers in the network. These active peers are obtained from one of the known Gnutella hosts such as gnutellahosts.com. The joining peer and its

neighbors can be many network hops away from one another. As another example, in Kazaa [27], a joining peer is assigned a randomly chosen super node. In both systems, peers providing a requested object could be far away (in terms of network hops) from the requesting peer, even though there could be other nearby peers who have that object. This means that the traffic may unnecessarily travel through many network hops and burden the backbone links. This is more of a concern in a P2P media streaming system, where objects are typically very large and consume more bandwidth to stream. Furthermore, large variability in the delay (i.e., delay jitter) yields poor quality of playback. Unlike current P2P systems, peers in our system are organized in a way that facilitates keeping the traffic as *local* as possible. Peers are divided into *clusters*. A cluster is a logical grouping of peers that are topologically close to each other.

Clustering of the web clients has been studied before in [4,3,14]. The main objective of web client clustering is to position proxy caches or server replicas close to the clients. Web clients clustering techniques are not directly applicable to our P2P streaming system, because they are either too coarse-grained [3] or too fine-grained [4,14]. The technique in [3] aggregates many autonomous systems (ASes) into one group. Thus a group would have too many peers and one super peer would not be able to handle all the traffic from them. On the other hand, the fine-grained techniques group together peers that either share the same network prefix [14] or use the same local DNS server [4]. Both techniques result in a large number of clusters, each likely having only 10s of peers. The problem then becomes managing the large number of small clusters.

We propose a new two-level clustering technique suitable for P2P systems, in which a cluster will have a moderate size that can be managed by a peer with reasonable resources. The applicability of the two-level clustering to P2P systems is validated by Internet statistics that we have gathered and analyzed. In the next section, we summarize the experiments performed and the findings that support the use of the two-level peer clustering. Then, in the remaining sections, we detail our clustering technique.

3.1. Internet statistics

The Internet is composed of numerous Autonomous Systems (ASes) interconnected together. Each AS has a 16-bit unique identifier, and is managed by a single administrative authority. Packets sent from a source to a destination across the Internet may travel through several ASes. Inter-domain routing protocols such as Border Gateway Protocol (BGP) [8] are used to construct the AS path that packets should take to reach the destination network. An AS is a collection of physical networks. Each network has a unique address. In TCP/IP networks with CIDR addressing [8], the network address (or network prefix) has two parts: an IP address and a length (or mask). For example, 128.10.0.0/16 is a network with up to (roughly) $2^{(32-16)}$ hosts. All hosts on this network have IP addresses with the same network prefix (128.10), i.e., the same leftmost 16 bits.

In this section, we try to gain insights on how peers are distributed over the Internet. Specifically, we would like to know how many peers will likely be on the same network and within the same AS. This will guide the organization of the peers in our proposed system. In the following sections, we explain the experiments that we have conducted on real Internet data and our findings.

3.1.1. Methodology of collecting statistics

First, we notice that the BGP routing tables provide a wealth of information that can provide the required statistics. A snapshot of a BGP routing table is shown in Fig. 8. Only three fields are shown: network prefix (the destination network), next hop (next router on the path to the destination network), and AS path (ASes on the path to the destination: the last one is the desti-

Prefix	Next Hop	AS Path
6.9.0.0/20	207.246.129.6	11608 2914 668 7170 1455
18.0.0.0/8	207.246.129.6	7018 1 3
128.2.0.0/16	207.246.129.6	11608 2914 5050 9
128.10.0.0/16	207.246.129.6	11608 2914 3356 1 19782 17

Fig. 8. A snapshot of the BGP routing table at router 207.246.129.6 in the autonomous system AS11608. Only the three fields relevant to the discussion are shown.

nation AS itself). We conduct the following steps to collect and analyze the data:

- *BGP data collection.* Routing tables from dozens of core BGP routers are collected. Each routing table provides a snapshot of the Internet from a different angle. Merging these routing tables together give a somewhat complete picture of the networks and the ASes in the Internet. We have collected this BGP data from the RouteViews Project [21]. The data we use is a dump of the BGP tables on May 1st, 2003.
- *BGP data processing.* The BGP data is in MRT (binary and compressed) format. We convert it to ASCII format using `route_btoa` command, which is a part of the MRT package [28]. Since the data is a merge of many BGP tables, it has a lot of redundancy. We have built scripts to extract the required fields (network prefix, length, AS number), remove the redundant data, and compute the required statistics. The cleaned data has 117,027 unique network prefixes in 14,950 different ASes.
- *Peer data collection.* IP addresses of some of the peers in one of the largest P2P systems (Gnutella) are collected. [1] The data file has 216,226 unique IP addresses.
- *Peer data processing.* We identify the network to which a peer is attached by a process similar to the IP lookup operation performed by IP routers. We have implemented a fast data structure called level-compressed trie [19,33] for this process. More details on this data structure are given in Section 3.2.4. For each IP address, we identify the network and the AS to which it belongs.

3.1.2. Observations

We summarize our observations in the following.

- *Network size*. Fig. 9 shows the distribution of the network size. The size is the maximum number of IP addresses a network can have, which is given by the network prefix length. For instance, a network with prefix length 24 (a traditional class C network) can have up to $2^{(32-24)} = 256$ addresses. The figure indicates that the current Internet has numerous small-size networks. These small networks have separate entries in the BGP tables.

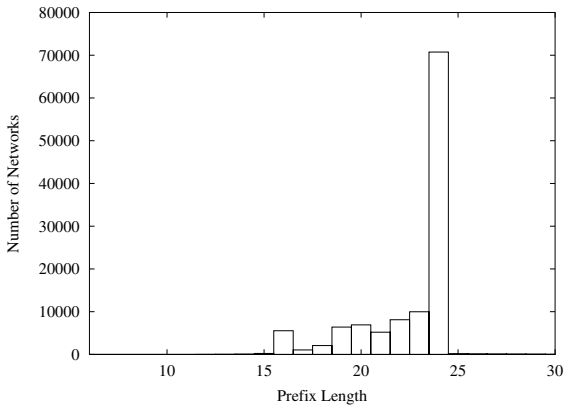


Fig. 9. Histogram showing the distribution of network sizes in the Internet. Many small (/24) networks.

- *Number of peers per network.* The total number of peers we have is 216,226. Those peers are distributed over 5,766 networks located in 2,013 ASes. Fig. 10 shows the cumulative distribution of the number of peers in each network. We notice that most of the networks have a small number of peers: 90% of the networks have less than 40 peers each, and the majority of them have up to 200 peers. This result matches the result in Fig. 9, which says that most of the networks are of small size.
- *Number of networks per AS.* Fig. 11 addresses the issue of how large an AS is. We count the number of unique networks in each AS and *rank* the ASes based on that number: the higher the number of networks in an AS, the lower its rank is. Fig. 11 shows that we have a very few ASes that have 500 or more networks each. The majority of ASes have less than 100 networks.
- *Number of peers per AS.* The cumulative distribution of the number of peers in each AS is shown in Fig. 12. The figure indicates that most of the ASes have less than 1000 peers each.

[1] We would like to thank Stefan Saroiu and authors of [34] for sharing this data with us.

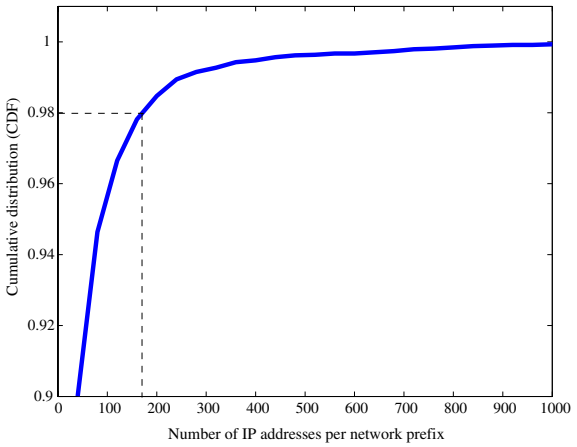


Fig. 10. Distribution of number of peers in each network. Ninety percent of the networks have less than 40 peers and almost all have less than 200 peers.

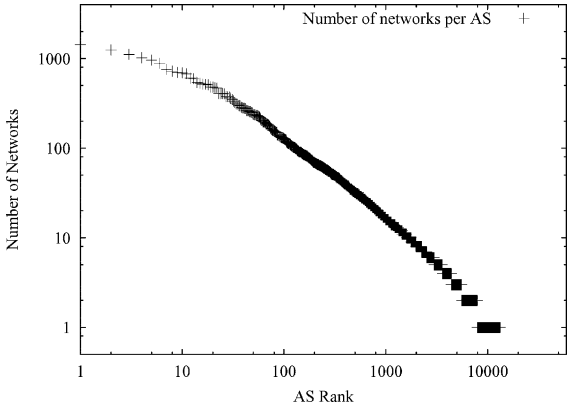


Fig. 11. Number of networks in each autonomous system (AS). The rank of an AS is based on the number networks it has: rank 1 means the AS that has the highest number of networks. The majority of ASes have less than 100 networks each. The average is 7.83.

3.1.3. Conclusion

Although these results do not cover all possible cases of peer distribution in the Internet, they give an idea on how this distribution might look like and therefore how peers can be grouped into clusters. When we organize peers into clusters we need to consider two issues: managing peers within each cluster and managing the clusters themselves. To manage peers within a cluster, a powerful enough peer should be selected. This peer is called a

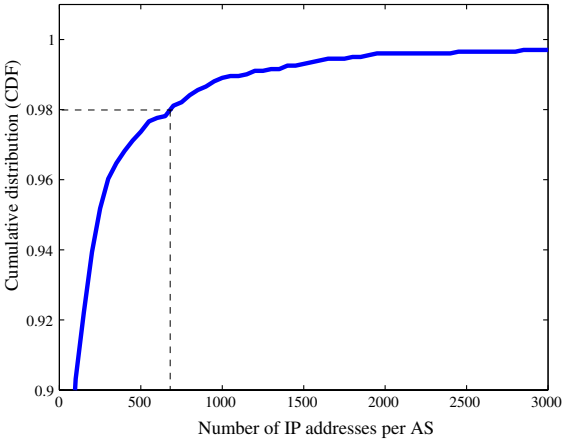


Fig. 12. Distribution of number of peers in each autonomous system (AS). Most of the ASes have less than 1000 peers each.

super peer. The capacity of the super peer should be proportional to the number of peers expected to join the cluster. For the system to function properly, clusters should be interconnected. Furthermore, an entity in the system should keep track of the current super peer of each cluster to direct a new peer to the correct cluster. There is a clear tradeoff: using few clusters with many peers each imposes less overhead in managing the clusters, but requires more powerful super peers to manage each cluster. Specifically, if we cluster peers based on their network attachment, we may end up with too many clusters, each with a small number of peers (10s of peers). This may impose significant overhead on the system. On the other hand, if we group all peers within an AS into one large cluster, we may not find a powerful enough super peer to manage the cluster. To strike a balance between the overhead imposed on managing clusters and the capacity needed to manage each cluster, we propose a two-level peer clustering approach.

3.2. Two-level peer clustering

We use two levels of aggregation to divide peers into clusters. Fig. 13 depicts the two-level peer clustering technique. In the first level, peers attached to the same network are grouped into the same network-level cluster, which we call a *network cluster* for short. The second level aggregates

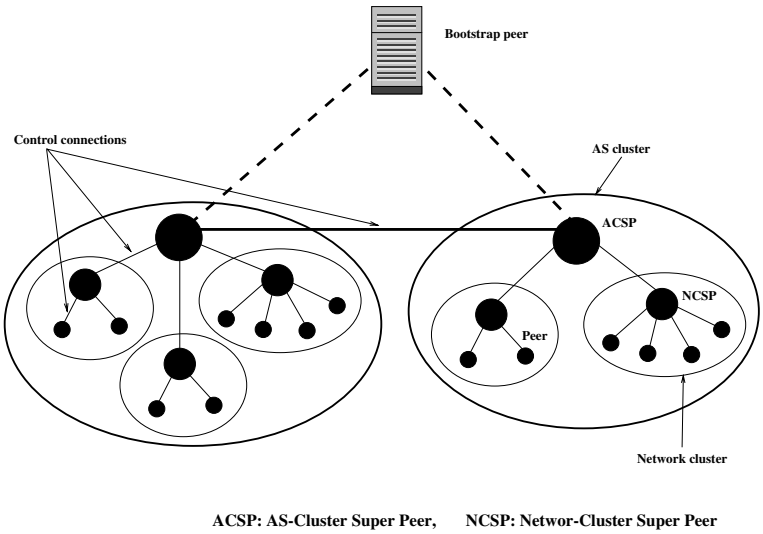


Fig. 13. Two-level clustering of peers.

all networks within the same AS into a larger cluster, called an *AS cluster*. Each cluster has a super peer, which performs special functions based on the cluster level. Notice that the responsibilities of a network cluster super peer (NCSP) are different from those of an AS-cluster super peer (ACSP). The super peer in a cluster is the most powerful peer willing to assume this role in that cluster. For robustness, a number of *backup* super peers are chosen. Details of maintaining the system in case of failures are given in later sections. In addition to the two levels, we assume that there is a set of *bootstrap* peers that are publicly known. A joining peer contacts one of the bootstrap peers, which will direct the joining peer to the appropriate cluster.

3.2.1. Peer interconnections

Two types of connections may exist among peers: data connection and control connection. The data connection is established between a supplying peer and a receiving peer. This connection carries the streaming data and uses the UDP transport protocol. The data connection is closed once the streaming session is over. The control connection is used to: (1) send control messages such as join, leave, and search, (2) transfer control information such as tables and indexes, and (3) exchange heartbeats to detect failures and maintain the system. The control connections use the reliable TCP protocol. The data connections are established on-demand, while the control connections exist as long as the peer is in the system. Fig. 13 shows the control connections among peers. Within a network cluster, each peer is connected to the super peer of that network cluster. All network-cluster super peers in an AS are connected to the ACSP of that AS. All ACSPs are connected to the bootstrap peers. Finally, control connections exist between an ACSP and some of its neighbor ACSPs.

3.2.2. Network cluster

Peers are grouped into network clusters based on their IP addresses and data gathered from a number of BGP routers. Peers with IP addresses that have the same longest prefix with one of the routing table entries are assigned the same network cluster. 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. This level of clustering is similar to the web clients clustering technique proposed in [14].

Each network cluster has a network-cluster super peer (NCSP). The NCSP maintains an index of the files available in its network cluster and their locations. The index is a small database with a limited query-processing facility. The index is kept current through the maintenance procedure discussed in Section 3.4.

3.2.3. AS cluster

In the second level of clustering, we group all networks located in the same AS into one AS cluster. Networks sharing the same AS number are typically under the same administrative authority and in many cases they are topologically close to each other. For example, all networks on a university campus will belong to the same AS. The objective of this AS clustering is to increase the number of peers that belong to the same logical cluster and therefore, reduce the total number of clusters in the system. Increasing the number of peers per cluster will increase the probability of locating the requested object within the cluster. The AS number of a network is identified using the AS Path field in the BGP routing table: the last AS on the path is the AS of the destination network. For example, in Fig. 8, the AS for the network 6.9.0.0/20 is 1455, and the AS for the network 128.10.0.0/16 is 17.

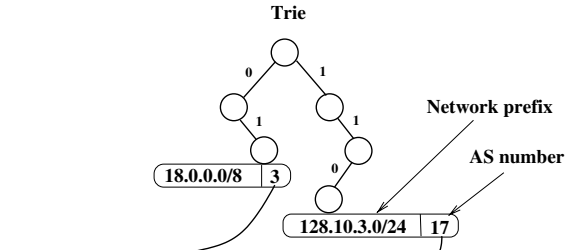
The AS-cluster super peer (ACSP) does not maintain an index to files in the AS, since this would impose a high load on it. Rather, it helps in two functions: identifying the network cluster of a new joining peer (see Section 3.3), and forwarding search queries to NCSPs within the same AS and to other ACSPs (see Section 4.1). To perform these functions, the ACSP maintains the data structure shown in Fig. 14. Since the number of network clusters per AS is not too large (on the order of 10s), a simple table is sufficient. The table contains for each network cluster a primary NCSP, a set of backups, and a timer. The timer is reset upon receiving a heart beat from the primary NCSP. If the timer expires, the first backup will become the primary NCSP.

Networkcluster	Primary NCSP	Timer	Backup NCSPs
128.10.3.0/24	128.10.3.10	15	128.10.3.133, 128.10.3.21
128.10.5.0/24	128.10.5.7	5	128.10.3.88, 128.10.3.2
128.10.6.0/24	128.10.6.25	30	128.10.3.210, 128.10.3.23
---	---	--	---

Fig. 14. Data structure maintained at each AS-cluster super peer (ACSP).

3.2.4. Bootstrap peers and initialization

The bootstrap peers direct new peers joining the system to the appropriate AS clusters. To do so, each bootstrap peer maintains the data structure shown in Fig. 15. The data structure has two parts: a trie and an AS table. Each entry in the AS table represents an AS cluster. It has the AS number, the primary and backup ACSPs for that cluster, and a timer. The trie is used to perform an operation similar to the IP lookup operation performed by IP routers. The IP lookup operation returns the entry in the routing table that shares the longest prefix with the input IP address. This longest prefix match operation is performed efficiently (in a few number of steps) using tries. In our system, we construct a level-compressed trie [33] from data gathered from dozens of BGP routing tables



Primary ACSP	Timer	Backup ACSPs
18.1.2.3	15	18.1.125.10, 18.10.3.21
128.2.5.7	5	128.2.3.88, 128.2.13.2
128.10.3.25	30	128.10.3.210, 128.10.3.23
---	--	---

Fig. 15. Data structure maintained by a bootstrap peer.

publicly available at [21]. A leaf node in the trie has two attributes: the network prefix, and the associated AS number. To find the corresponding AS cluster of a peer, the trie is used to find the leaf node whose network prefix attribute shares the longest prefix with the IP address of that peer. The AS attribute of the best matching node has a pointer to the corresponding entry in the AS table. The construction of the trie is done only once at each bootstrap peer.

Note that the dynamic nature of the BGP routing tables does not affect the clustering technique, because the clustering technique uses only the network prefix and the AS number entries. Although the path to a network may change, the network address and the AS number do not. Even if a network prefix entry is dropped from one BGP table, it is unlikely that this prefix will be dropped from all BGP routing tables used in the construction at the same time. We need, however, to update the trie when a new AS is created, which happens on a much larger time scale than changes in BGP routes. Therefore, the bootstrap peers may need to update the trie on the scale of days or even weeks. This update is not expensive. It involves downloading the current BGP routing tables (about 2 MB) from [21] and checking for newly added networks and ASes.

At the very beginning (no peers yet), the AS table is empty and the trie is constructed a priori. When the first peer joins, the bootstrap peer determines its AS cluster using the trie. This peer becomes the ACSP for this cluster and its data structure is initialized with the network prefixes that fall within this AS cluster. The bootstrap peers can easily compute all network prefixes that have the same AS number using the trie. If another peer joins with the same AS number, the most powerful of the two will become the ACSP. If it falls within another AS, it will become the ACSP of the new AS cluster. More details on peer joining and leaving are given in Section 3.3.

3.3. Peer join and leave

Peer join. To join the system, a peer contacts one of the bootstrap peers. The bootstrap peer identifies the AS cluster of the joining peer and replies with a short message. The message contains the primary ACSP and its backups of the cluster. The joining peer sends a join request to the primary ACSP. The ACSP determines the network cluster of the joining peer and returns a list of NCSPs (primary and backups). The joining peer establishes a control connection with the primary NCSP. Over this control connection, the joining peer sends information about the locally cached files and its level of cooperation with the system. The level of cooperation is specified in terms of three resources: bandwidth, CPU, and storage. For example, a peer specifies the bandwidth as: "Ethernet, 10 Mb/s, 30%". This means that the peer is connected to the Internet through a 10 Mb/s Ethernet and is willing to share up to 30% of this bandwidth with the system. The primary NCSP updates its index with the new peer and its data. Note that the backup ACSPs (NCSPs) are included in the messages to shorten the joining time in case the primary ACSP (NCSP) has failed and the bootstrap peer has not detected the failure yet. In this case, if the joining peer gets no response from the primary ACSP (NCSP), it tries the first backup ACSP (NCSP), then the second, and so on.

A joining peer may be promoted to a super peer (NCSP, ACSP, or a backup for either of them), if it offers more resources than the current super peer. After the joining process finishes, the promotion process starts from bottom up. The NCSP responsible for the new peer compares the new peer's resources versus its own and each of the backups' resources. Although all resources (bandwidth, CPU, and storage) can be used in the comparison, we use the most important one: offered outbound bandwidth. This is because fast CPUs and large disks are abundant nowadays, and unlikely to become the bottleneck resources in the super peer. A *threshold* is used in the comparison to prevent frequent change of super peers. For example, if the new peer's bandwidth exceeds the current super peer's bandwidth by at least 25%, the new peer will replace the current super peer. If the new peer becomes the primary NCSP, it gets the index from the replaced NCSP and informs the parent ACSP, all backup NCSPs and peers in the network cluster. The parent ACSP performs a

similar comparison to determine whether the new peer can become an ACSP. If the comparison succeeds, the new peer is promoted to an ACSP, and it informs the bootstrap peers, the backup ACSPs, and other ACSPs that have connections with the replaced ACSP.

Graceful peer leave. Graceful means that the peer leaves the system through the normal way: closing the P2P software agent, which triggers sending leave messages to the appropriate peers. Peer failure is presented in the next section. Three different cases need to be considered: regular peer leave, NCSP leave, and ACSP leave. When a regular peer leaves the system, it sends a leave message to its NCSP, which updates the index to reflect this leave. No further actions are needed. Before leaving the system, an NCSP first selects a new primary NCSP from the backup list, informs it, and updates its index. Then, it sends a leave message to its parent ACSP and to the active peers in the network cluster. The leave message contains the IP address of the new NCSP. The new NCSP establishes a control connection with its parent ACSP. The active peers close the connection with the old NCSP and open new ones with the new NCSP. In a similar way, when an ACSP leaves, it selects a new one, and informs the bootstrap peers and all NCSPs within the AS cluster.

3.4. Peer failures and reliability

This section describes how the system detects and recovers from peer failures. By failure we mean that a peer leaves the system suddenly without any notification. We consider only fail-stop failure model, in which a peer stops sending and receiving messages once it fails. Other failure models (e.g., Byzantine) are outside the scope of this paper.

Heartbeats. Periodic "I am alive" control messages (heartbeats) are exchanged among peers over the control connection in the following manner: each peer sends heartbeats to its *parent* and to its backups, if there is any. Specifically, peers in a network cluster send heartbeats to the NCSP of the cluster. These heartbeat messages are tiny messages with only four bytes of data to identify the message. Each NCSP within an AS cluster sends heartbeats to its parent, which is the ACSP of the cluster. Moreover, the NCSP sends heartbeats to its backups. Similarly, an ACSP sends heartbeat messages to its parent (the bootstrap peers) and its backups. The heartbeat messages sent by ACSPs and NCSPs are a bit longer: they contain the IP addresses of the backups sorted based on their capabilities. That is, the first IP in the list is the most powerful backup, and therefore will be the first candidate to replace the super peer in case of failure.

Failure detection. Since heartbeats are short messages and sent reliably over the control connection, losing one or two of them indicates sender failure. In our system, if two successive heartbeats are not received, the sender is assumed to have failed. The reaction to the failure depends on the type of the failed peer. If it is a regular peer (i.e., neither ACSP nor NCSP), its record in the NCSP index is removed. On the other hand, if the failed peer is an NCSP, the first peer in the backup list becomes the new NCSP. Note that, since all backups and the parent ACSP were receiving the same heartbeats from the failed NCSP, they will all agree on the same new NCSP. The new NCSP establishes a control connection with its parent ACSP. The new NCSP notifies the active peers in the network cluster about the failure. The active peers are those listed in the index table maintained by the NCSP and replicated at the backups. The index table at the backups is frequently updated by the primary NCSP. The active peers then open new connections with the new NCSP. Similar steps are taken in case of ACSP failures.

Overhead. Heartbeats are short messages sent over relatively long periods (30–90 s). We estimate the overhead on an NCSP as follows. Assume there are 200 peers in the network cluster and the heartbeat period is 60 s. The message carrying the heartbeat from a regular peer to the NCSP has a size of 44 bytes (20 bytes for IP header + 20 bytes for TCP header + 4 bytes for data). Therefore, the inbound traffic overhead due to receiving heartbeats is: $(200 \times 44 \times 8)/60 = 1.17$ Kb/s. Every incoming heartbeat is acknowledged (since it is TCP) using a message of size 40 bytes (no data, just the ACK bit is set in the TCP header). Moreover, the NCSP sends out heartbeats to its

backups and its parent. These heartbeats contain the IP addresses of the backups. Assume we have two backups. Then, the message size is 52 bytes. Therefore, the outbound traffic overhead is: $(200\times40\times8)/60+(3\times52\times8)/60=1.09$ Kb/s. From this simple calculation, we see that the overhead is a very small fraction of the super peer bandwidth. It is less than 0.12% for a super peer connected to 10 Mb/s Ethernet.

4. Cluster-based searching and dispersion

4.1. Searching

As discussed in Section 3, peers in the system are organized in a network-aware fashion. This organization enables the searching algorithm to locate *nearby* peers who have segments of the requested media file. The proposed cluster-based searching algorithm can be summarized in the following steps (consider Fig. 13):

1. The requesting peer sends a lookup request to its own network-cluster super peer (NCSP). The NCSP has an index of the files available in the network cluster and their locations. The NCSP checks whether there are enough suppliers to serve the new request. If there are enough suppliers, the result is returned to the requesting peer in the form: $\langle j, \mathbb{P}^{(\text{main})}, \mathbb{P}^{(\text{backup})}\rangle$, where j is the segment number, $\mathbb{P}^{(\text{main})}$ is the set of main suppliers, and $\mathbb{P}^{(\text{backup})}$ is the set of backup suppliers. Each element in the supplier sets has two fields: the IP address of the peer and the offered rate by that peer.
2. If there are no sufficient suppliers in the network cluster, the NCSP forwards the *remaining* part of the request to its own AS-cluster super peer (ACSP). The ACSP does not maintain an index of files, but it does have pointers to other NCSPs in the same AS cluster. The ACSP sends the request to the NCSPs one at a time until sufficient suppliers are accumulated. The result is then returned to the requesting peer through its NCSP.
3. If there are no sufficient suppliers in the AS cluster, the ACSP forwards the remaining part of the request to one of its neighbor ACSPs, which in turn sends the request to NCSPs in its cluster. The results are accumulated and returned to the requesting ACSP then to the NCSP and finally to the requesting peer.

Comments. In the first step, enough suppliers means that the main suppliers can provide all segments at the full rate. To maintain the full quality in case of peer failures, the system chooses a number of backup suppliers. In the second and third steps, the *remaining* part of the request refers to the segments that were not found with the full rate in the previous step. To reduce the delay that might result in the second and third steps, the request can be sent to multiple super peers simultaneously. After choosing the suppliers, the indexes of the involved NCSPs are updated by marking the entries of the corresponding suppliers as busy for the duration that these suppliers are scheduled to stream. Note also that the index is updated when a peer fails, joins, or leaves as discussed in Sections 3.3 and 3.4. Finally, if the system does not have sufficient peers to satisfy the request, an empty supplier list is sent to the client. The client then backs off and tries after an exponentially increased waiting time.

4.2. 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 in each cluster to serve all expected 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. 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. Two dispersion algorithms are proposed: a simple *random* dispersion algorithm, and a *cluster-based* dispersion algorithm. We describe the two

algorithms in this section and evaluate them in Section 5.3.

4.2.1. Random dispersion

The simplest way for a peer P_y to cache N_y segments is to randomly choose them. This is simple in the sense that no communication is needed with the super peers. However, it does not ensure that, on the average, the same number of copies of each segment is cached. Given that all segments are equally important for streaming the entire file, this is a critical issue. 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.

4.2.2. Cluster-based dispersion

The cluster-based dispersion dynamically adjusts the available capacity within each cluster according to the average number of client requests from that cluster. The `ClusterDisperse` algorithm, shown in Fig. 16, is to be run by the network-cluster super peers (NCSPs). Consider one media file with N segments, rate R Kb/s, and duration T hours. The algorithm requires maintaining 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 NCSP 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, $0 \leqslant u_x \leqslant 1$, the fraction of time P_x is online. Recall that the peer is not available all the time. For every network cluster $\mathtt{c}$, the NCSP maintains the following: (1) L_s, $1 \leqslant L_s \leqslant N$, the next segment to cache; (2) $q_{\mathtt{c}}$, the average request rate (per hour) the media file is being requested by clients from $\mathtt{c}$. $q_{\mathtt{c}}$ represents the required capacity in the cluster $\mathtt{c}$ per hour; (3) $a_{\mathtt{c}}$, the average number of copies of the movie cached by peers in cluster $\mathtt{c}$. $\mathtt{c}$ 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 Eq. (1) computes the effective number of copies available in the cluster. It takes

ClusterDisperse Algorithm

```
1. /* Executed by the network cluster super peer (NCSP) of cluster c */
2.     L_s ← 1 /* Initial value */
3.     while TRUE do
4.         Wait for a caching request
5.         /* Got request from peer P_y to cache N_y segments with rate R_y */
6.         Compute a_c, q_c and get A, Q from the ACSP
7.         if q_c > a_c or Q ≫ (1/T)A then /*need to cache in round robin */
8.             if (L_s + N_y − 1) ≤ N then
9.                 L_e = L_s + N_y − 1
10.            else
11.                L_e = N_y − (N − L_s + 1)
12.            end if
13.            Send reply to P_y /*Peer P_y caches from segment L_s to segment L_e*/
14.            L_s = L_e + 1
15.        end if
16.    end while
```

Fig. 16. Pseudo-code for the cluster-based dispersion algorithm.

Table 1
Symbols used in the ClusterDisperse 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_c	Next segment to cache in cluster c
	a_c	Average number of copies of the movie cached by peers in cluster c
	q_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)

into consideration the limited capacity and heterogeneity of peers, since peers do not offer server-like behavior as shown in [34]. The equation accounts for two facts. First, peers are not always online through the term u_x, which is the fraction of time peer P_x is available. Second, peers do not cache all segments at the full rate through the term R_xN_x/RN. Dividing a_c by T results in the number of requests that can be satisfied per hour, since every request takes T hours to stream. Hence, $(1/T)a_c$ represents the available capacity in the cluster c per hour.

Two global variables are maintained: (1) $A = \sum_c a_c$, the average number of copies of the movie cached by all peers in the system. (2) $Q = \sum_c q_c$, 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. The computation of these global variables are propagated from NCSPs to bootstrap peers and aggregated at ACSPs, as follows. Every NCSP of a network cluster c periodically (on order of minutes) reports its a_c and q_c to its own AS cluster super peer (ACSP). The ACSP aggregates these values from all network clusters in the AS and reports periodically (on the order of 10s of minutes) to the bootstrap peer. Computing and exchanging these variables impose a little overhead on the peers because: (1) the computation is a simple sum and done infrequently, and (2) the communication is a short control packet which could be a part of the heartbeats exchanged between peers (Section 3.4). After the up to date values of A and Q are computed, they are sent to all ACSPs.

The algorithm proceeds as follows. Upon getting a request from peer P_y to cache N_y segments, the NCSP computes a_c, q_c [2] and gets A, Q from its ACSP. 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_s 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_s, 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_s is updated to 2, and so on. This ensures that we do not over cache some segments and ignore others. Furthermore, the ClusterDisperse 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 $(1/T)A$,

[2] 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.

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.

5. Evaluation

We evaluate the performance of the proposed architecture through extensive simulation experiments. We use the Network Simulator *ns-2* [20] in the simulation. Three sets of experiments are presented. The first set of experiments (Section 5.1) addresses the system-wide performance parameters such as capacity and waiting time as well as how peers' levels of cooperation affect these parameters. The second set of experiments (Section 5.2) focuses on the client-side performance parameters such as the buffer size and the effect of peer unreliability on the quality of playback. The third set of experiments (Section 5.3) evaluates the proposed cluster-based dispersion algorithm and compares it with the random dispersion algorithm.

In all of the experiments, we use large hierarchical, Internet-like, topologies. Fig. 17 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 at 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 [5]. We then probabilistically add dial-up (e.g., DSL and Cable modem) and LAN hosts to routers in the stub domains. The details of the topology (e.g., number of routers, number of peers, and number of domains) may differ from an experiment set to another in order to emphasize the performance parameters being studied in that set. However, all topologies have the structure shown in Fig. 17.

The results presented in the following sections are averages over many runs each with a different seed for the pseudo-number generator used in *ns-2*.

5.1. System-wide performance

We study the performance of the system 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 time and the level of cooperation changes:

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 getting the media file;

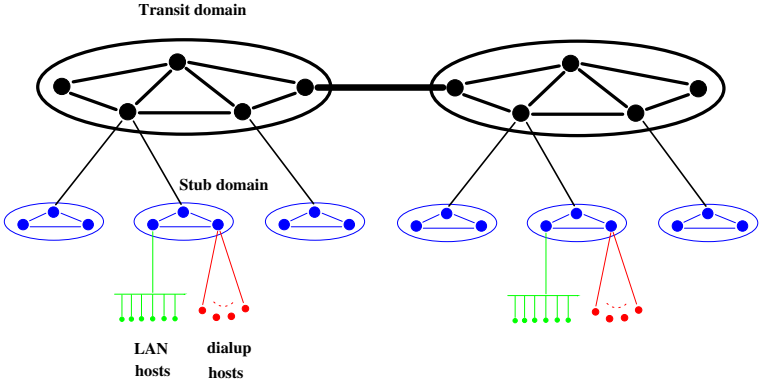


Fig. 17. Part of the topology used in the simulation.

3. the average number of satisfied (or rejected) requests; and
4. the load on the seeding peer.

The topology used in this set of experiments consists of 20 transit domains, 200 stub domains, 2100 routers, and a total of 11,052 hosts distributed uniformly at random.

5.1.1. Simulation parameters and scenario

We have the following fixed parameters:

1. A media file of 20 min duration recorded at a CBR rate of 100 Kb/s and divided into 20 one-minute segments.
2. The dial-up 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 peer 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 four 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 2 min.

We vary the caching percentage from 0% to 50% and study the system under various client arrival patterns. 0% caching means that the requesting peer does not store any segment of the media file; whereas with 50% caching, it stores half of the file. The results are summarized in the following sections.

We simulate the following scenario. A seeding peer 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 protocol described in Section 2.2 is applied. 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.

5.1.2. Results for constant rate arrivals

Fig. 18 shows the behavior of the P2P architecture when the constant rate arrival pattern shown in Fig. 18a is applied to the system. Fig. 18c shows how the system's capacity evolves over time. The average service rate, increases with the time, because as 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 a high caching percentage (i.e., higher level of cooperation from peers). For instance, with 50% caching, the system is able to satisfy all the requests submitted at 5 requests/min after about 250 min (about 4.2 h) from the starting point. We can use Fig. 18c 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 (in which all clients are served)? To illustrate, suppose that the target service rate is 2 requests/min. Then, 30% caching will be sufficient and the steady state will be achieved within less than 5 h. The average waiting time, shown in Fig. 18b, is decreasing over time, even though the system has more concurrent clients, as shown in Fig. 18d. This is due to the rapid amplification of the capacity. Fig. 18d complements Fig. 18c by showing that the average rejection rate is decreasing over the time.

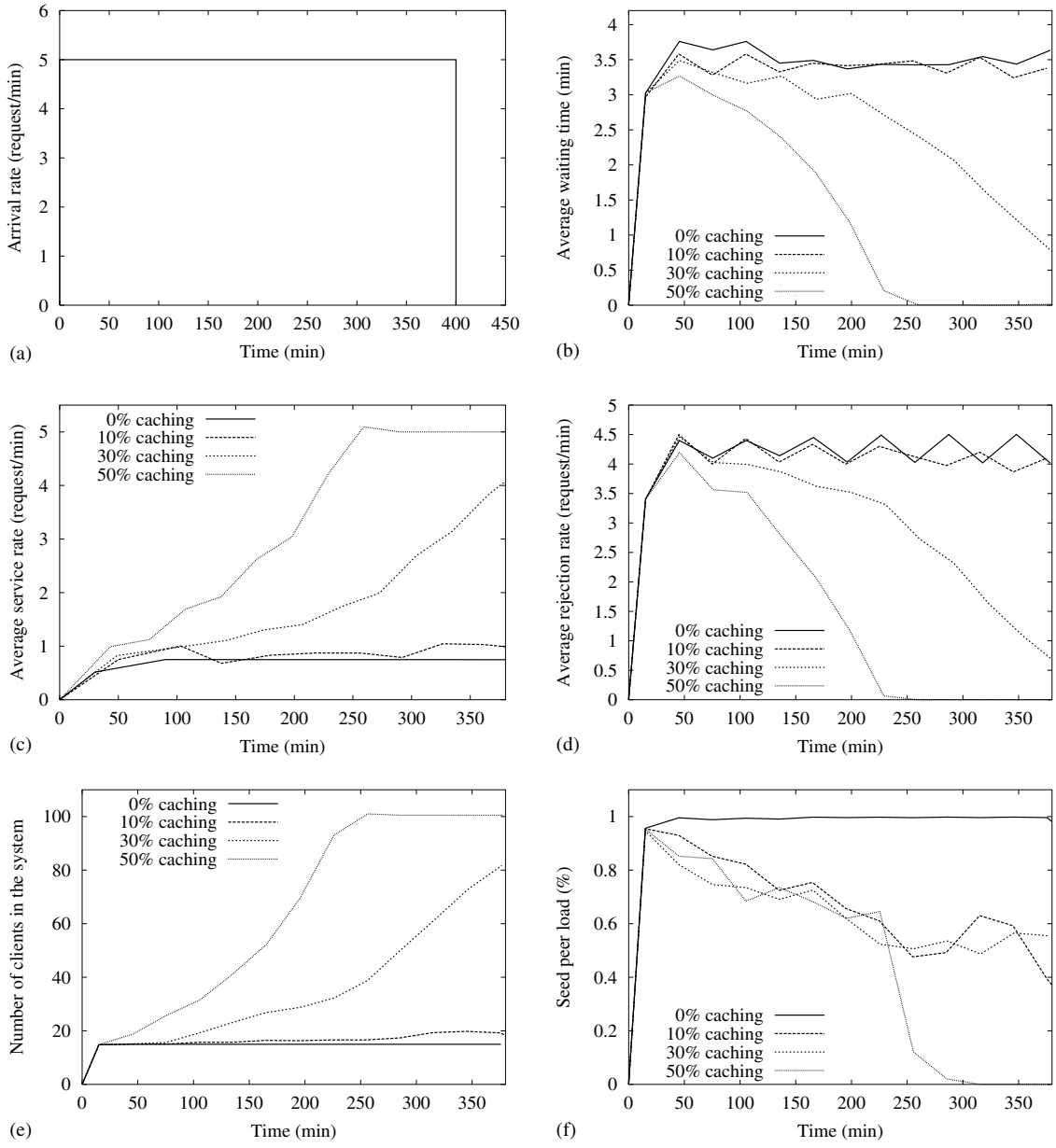


Fig. 18. Performance of the P2P architecture under constant rate arrivals: (a) clients arrival pattern: constant rate arrivals; (b) average waiting time; (c) average service rate; (d) average rejection rate; (e) number of clients concurrently being served; and (f) load on the seeding peer.

Finally, Fig. 18e and f verify the diminishing role of the seeding peer. 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 peer decreases over the time, especially with high caching percentages. For instance, with 50% caching and after about 5 h, we have 100

concurrent clients, i.e., 6.7 times the original capacity, and none of them is served by the seeding peer. Reducing the load on the seeding peer is an important feature of the architecture, because it means that the seeding peers need not to be powerful machines with high network connectivity. Besides being moderate machines, the seeding peers are used only for a short period of time. Therefore, the cost of deploying and running these seeding peers (in case of a commercial service) is greatly reduced.

5.1.3. Results for flash crowd arrivals

Flash crowd arrivals are characterized by a surge increase in the client arrival rates. These kinds of arrival pattern arise in cases such as the release of a popular movie or a publicly interesting event. To simulate the flash crowd arrivals, we initially subject the system to a small request rate of 2 requests/min for some period of time (*warm up* period), and then suddenly increase the arrival rate 10 folds to be 20 requests/min for a limited time (100 min). The arrival pattern is shown in Fig. 19a.

The results shown in Fig. 19 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 Fig. 19c, reaches as high as the client arrival rate (i.e., 20 requests/min) during the crowd period. Therefore, the system does not turn away any customers, when the caching percentage is 50%, as shown in Fig. 19d. Moreover, all customers are served without having to wait, as shown in Fig. 19b.

Fig. 19e indicates that, during the crowd period and with 50% caching, 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, Fig. 19f shows that none of the clients is being served by the seeding peer, which confirms that the seeding peer'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 peer.

5.1.4. 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/min (i.e., mean inter-arrival time of 0.1 min) are shown in Fig. 20. Notice that, Fig. 20a shows the density functions of the inter-arrival time distribution (exponential distribution). The results are similar to the case of constant rate arrivals, except that 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.

5.2. Client-side performance

This section studies the initial buffering needed by the client and the effect of supplying peer switching on the quality of playback. We use the number of "glitches" or pauses during the streaming session to quantify the quality of playback. More pauses means poorer quality. For a smooth playback, we should not have any pauses. This is a rather stringent quality measure, since we either play the segment or pause till we get all its packets. No error concealment or interpolation techniques are used.

5.2.1. Simulation parameters and scenario

In this set of experiments, we set the streaming rate to be 512 Kb/s, and we stream a file of 34 min duration. The file is divided into 1024 segments, each of 2 s length. The simulated topologies have 2,000 nodes. We vary the initial buffering time from 0 to 20 s. We measure the average number of times the client experiences buffer underflow for the simulated initial buffering. Every buffer underflow instance causes a *pause* in the playback until sufficient data packets arrive. These pauses are mainly due to supplier *switching*. A switching happens if the rate coming from a supplying peer decreases due to failure or congestion. The supplying peer may also stop sending because it has no more segments to send (recall that peers do not cache the entire file). To simulate switching, we populate peers with only 25% of the segments chosen at random and we allow peers to fail. When switching happens, we delay sending packets from

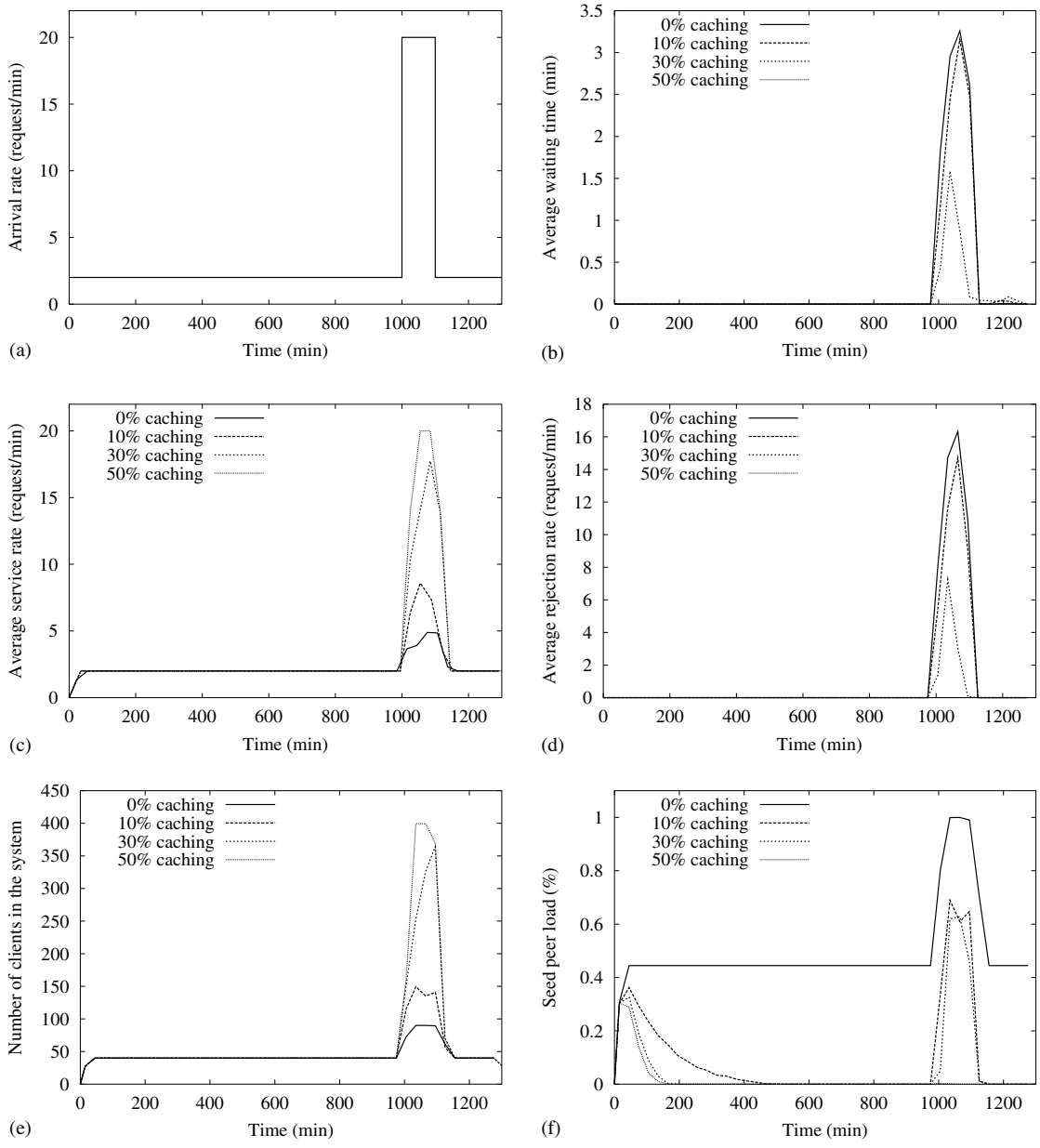


Fig. 19. Performance of the P2P architecture under flash crowd arrivals: (a) clients arrival pattern: flash crowd arrivals; (b) average waiting time; (c) average service rate; (d) average rejection rate; (e) number of clients concurrently being served and (f) load on the seeding peer.

the replacement peer(s) by a random time between 0 and 1 s. This is called the *switching time*, during which the degraded peer is detected and a replacement peer is notified.

To simulate playback of the media, an independent playback process is scheduled at regular times. The first call of this process is after the simulated initial buffering time. Then, it is called

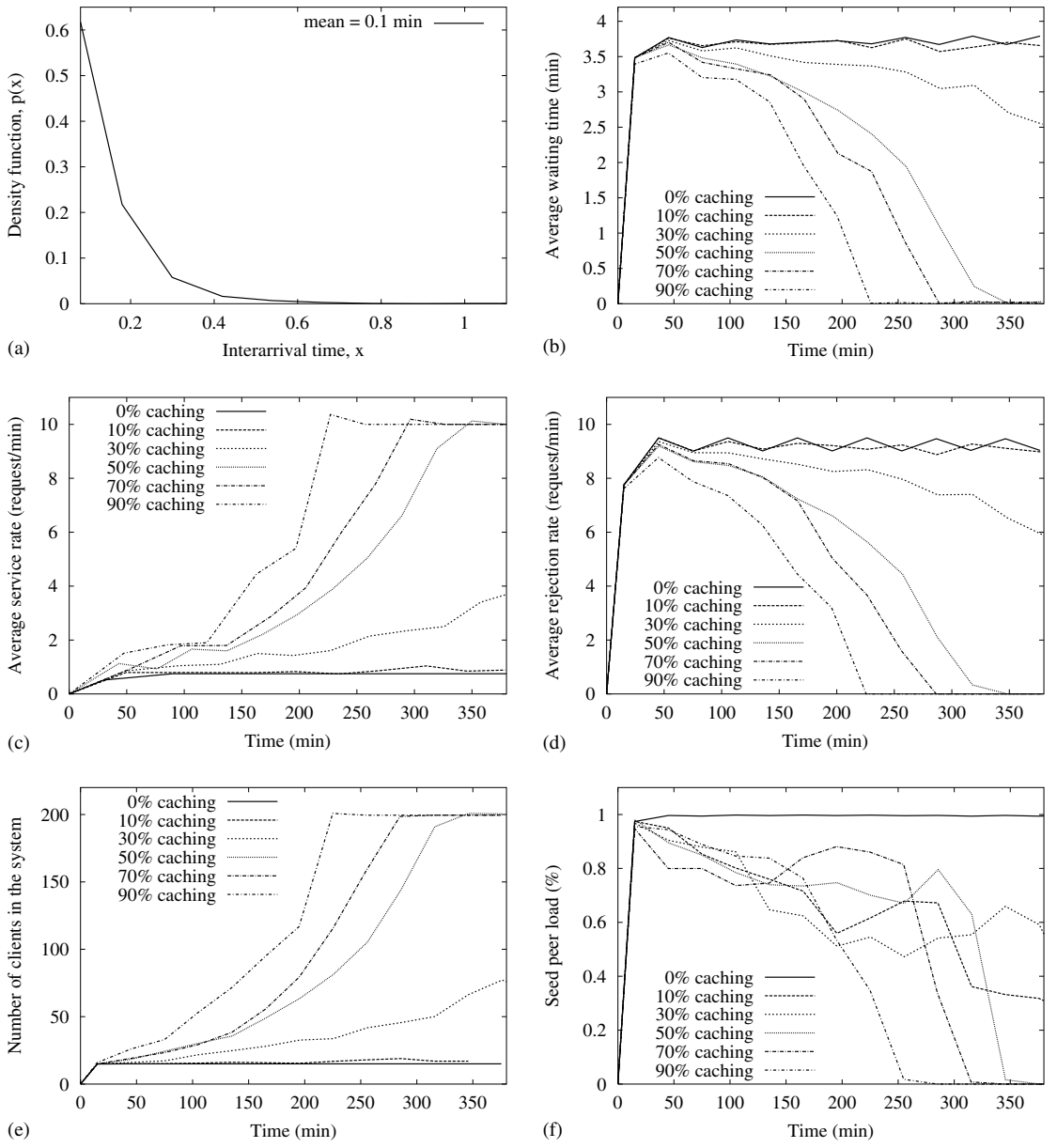


Fig. 20. Performance of the P2P architecture under Poisson arrivals, mean inter-arrival time = 0.1 min: (a) density function of the inter-arrival time (exponential with mean 0.1); (b) average waiting time; (c) average service rate; (d) average rejection rate; (e) number of clients concurrently being served and (f) load on the seeding peer.

every 2 s (the segment length in this experiment). When the playback process is invoked to play segment i, it checks the buffer for all packets belonging to segment i. These packets are identified through their sequence numbers. If all packets are available, the playback of segment i is successful and the playback process is scheduled for segment $i+1$ after 2 s from the current simulation time. If any packet is missing, a pause is encountered. The playback process is scheduled for the

same segment after the pause time, which is one second in this experiment.

We simulate the following scenario. A value for the initial buffering time is set. A client is chosen at random. The set of suppliers (main and backup) are constructed for that client. The streaming starts from the set of main suppliers. Switching happens at random times and replacement peers are chosen from the backup list. On average, 17 switching events occur during each session. After the initial buffering period, the first invocation of the playback process is scheduled. We count the number of pauses encountered throughout the session. After the streaming session is over, the experiment is repeated for another randomly chosen client. We simulate 10 different sessions. We compute the minimum, maximum, average number of pauses over these 10 sessions. Then, another value for the initial buffering is set and the whole scenario is repeated again.

5.2.2. Results

Fig. 21 shows the effect of switching on the quality of display for different values of the initial client buffering. For a small initial buffering of two seconds, we expect an average of 9 pauses and a maximum of 12 pauses. A buffer size of 12 s or more will absorb all transient effects during supplier switching. We believe that 10–20 s of buffering is a reasonable requirement to ensure full quality for streaming a half-hour movie at a rate of 512 Kb/s. Note that this buffering is meant to cover the peer switching introduced by the limited reliability of peers. No prolonged congestion periods were simulated in this experiment. Recall that the proposed architecture strives to serve each request from the local cluster, in which congestion is unlikely to occur (congestion usually occurs at the peering points between Internet service providers).

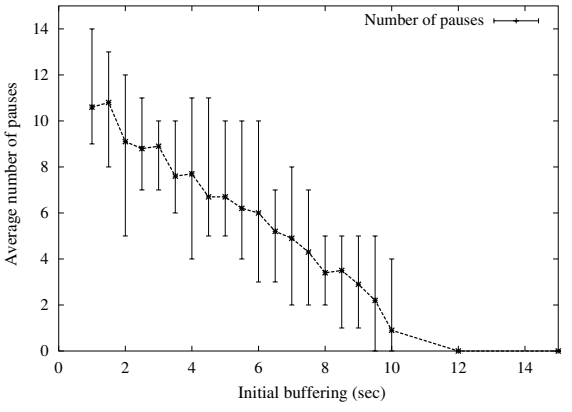


Fig. 21. Effect of switching on the quality of playback. A reasonable client initial buffering is required to tolerate the dynamic nature of suppliers in the proposed architecture. Note: mid-point is the average number of pauses, while the lower, and higher points are the minimum and maximum number of pauses, respectively.

5.3. Evaluation of the dispersion algorithm

We evaluate the proposed cluster-based dispersion algorithm and compare it against the random dispersion algorithm. We evaluate the efficiency of the dispersion algorithm by measuring the average number of network hops traversed by the requested stream. A smaller number of network hops indicates savings in the backbone bandwidth and less susceptibility to congestion, since traffic passes through fewer routers. Note that the comparison criteria not only depend on the dispersion algorithm but also on the searching algorithm. We use our cluster-based searching algorithm in the experiments. Therefore, we in fact evaluate both the searching and the dispersion algorithms together.

5.3.1. Simulation parameters and scenario

In this set of experiments, most of the simulation parameters are the same as in Section 5.1, except that the topology is larger. The topology has 100 transit domains, 400 stub domains, 2400 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 percent-

ages, 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/min.

The simulation scenario is similar to the scenario in the first set of experiments (Section 5.1) with one difference in 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 `ClusterDisperse` algorithm, which caches the same number of segments but selects them carefully. Each experiment lasts for 500 min 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 sections.

5.3.2. Results for 5% caching

Fig. 22a shows the pmf of the number of network hops for both the random and the `ClusterDisperse` dispersion algorithms. The pmf curve of the `ClusterDisperse` algorithm is shifted to the left of the random algorithm. This indicates that the traffic crosses a fewer number of hops using the `ClusterDisperse` 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 `ClusterDisperse` 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, Fig. 22b, 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.

5.3.3. Results for other caching percentages

Similar results were obtained for other caching percentages, as shown in Figs. 23 and 24. The main observation is that the difference between the two algorithms 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.

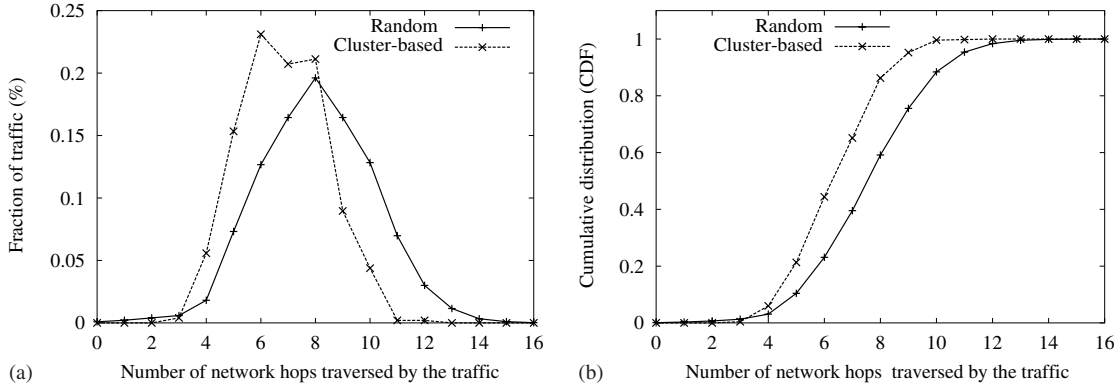


Fig. 22. Comparison between the random and the cluster-based dispersion algorithms, 5% caching: (a) probability mass function (pmf) of the number of network hops and (b) cumulative distribution function (CDF) of the number of network hops.

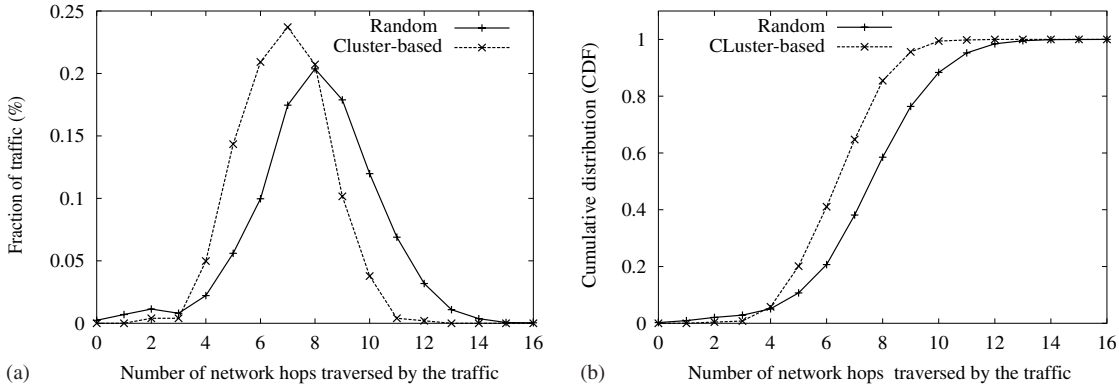


Fig. 23. Comparison between the random and the cluster-based dispersion algorithms, 10% caching: (a) probability mass function (pmf) of the number of network hops and (b) cumulative distribution function (CDF) of the number of network hops.

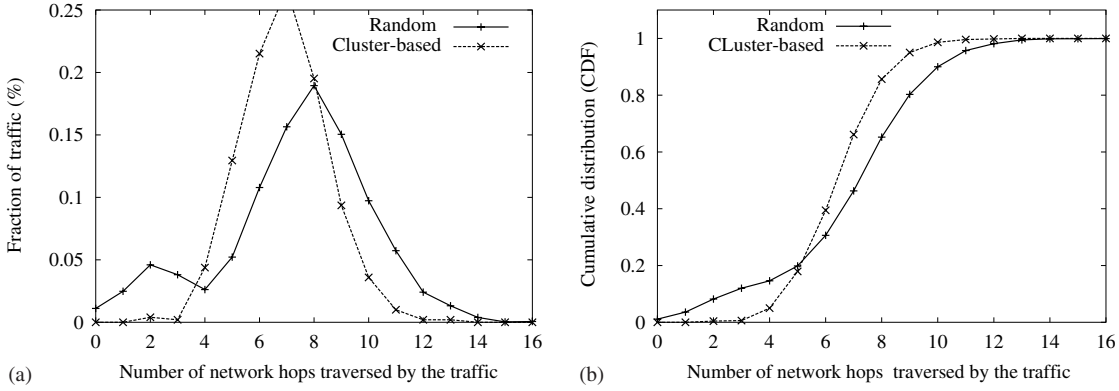


Fig. 24. Comparison between the random and the cluster-based dispersion algorithms, 30% caching: (a) probability mass function (pmf) of the number of network hops and (b) cumulative distribution function (CDF) of the number of network hops.

6. Related work

We summarize related work in both areas of P2P and media streaming systems.

6.1. P2P systems

In the last few years, 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 replication), and research on building P2P systems. Searching (or routing) protocols such as CAN [30], Chord [37], Pastry [31], and Tapestry [43] 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 (except for Pastry) they do not explicitly consider network locality. Other searching techniques do not provide such guarantees but they support flexible queries [42]. Related to our dispersion algorithm are the efficient replication strategies proposed in [6], which minimize the expected search size. These strategies are *proactive* in nature in the sense that files may be replicated at hosts that did not request them. Our dispersion algorithm is different, a peer may only

cache whatever it has requested. Given that media files are typically large, proactively replicating them imposes a large overhead on peers. Besides, peers are not always willing to cache.

On the systems side, Gnutella [26] and Kazaa [27] are the largest currently running file-sharing systems, after legal problems brought down Napster [29]. Freenet is another file-sharing system focusing on the *anonymity* of both the producer and consumer of the files [25]. Examples of large-scale storage systems built on top of P2P architectures are presented in [9,15], and [32]. Our proposed system adds one more to the list but with a new service, namely, media streaming.

6.2. Media streaming

Significant research effort has addressed the problem of efficiently streaming multimedia, both live and on demand, over the best-effort Internet. In the client/server paradigm, 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 [1] and Digital Island [24] follow similar approaches to provide media streaming and other services. Our approach does not require powerful proxies or caches. Rather, it uses peers' extra resources as numerous tiny caches. These tiny caches do not require large investments and collectively enlarge the capacity of the system in a way that potentially outperforms any powerful centralized caches.

The distributed video streaming framework [18] is 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 supplying peers decide on the rates at which they are willing to provide.

Related to our work are systems like *SpreadIt* [10] for streaming live media and *CoopNet* [22,23] for both live and on-demand streaming. Both systems build distribution trees using application-layer multicast and, like ours, they rely on cooperating peers. 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. *CoopNet* [23] also supports on-demand media streaming; a client first contacts a server, which redirects it to peers that recently received the movie. The authors assume that a peer can (and is willing to) support the full rate required for streaming and they do not address the issue of quickly disseminating media files into the system.

Similar to our architecture, a hybrid media streaming system is proposed in [40]. Unlike us, the authors assume that a peer stores all segments of the media file and is "always on". In contrast, we account for peers' limited capacity and availability. They do not discuss dispersion algorithms and assume a requesting peer will always cache the entire file. Under the above assumptions, the authors present a neat analytic analysis of the *handoff* point at which all clients are served by peers. These systems do not take network locality into account when selecting the supplying peers. Finally, in our previous work [41], assuming peers cache the entire file, we have proposed a differentiated admission control algorithm in a purely distributed P2P media streaming architecture. The algorithm prefers admitting peers with higher out-bound bandwidth and decreases the average waiting time and buffering delay for all clients.

7. Conclusions and future work

We presented a hybrid architecture for on-demand media streaming that can serve many clients in a cost effective manner. We presented the details of the architecture and showed how it can be deployed over the current Internet. Specifically, we presented the streaming protocol used by a participating peer to request a media file from the system; a cluster-based dispersion algorithm, which efficiently disseminates the media files into the system; and a cluster-based searching algorithm to locate nearby peers who have segments of the requested media file. Through a large-scale

simulation study, we showed that our architecture can handle several types of client arrival patterns, including suddenly increased arrivals, i.e., flash crowds. Our simulation results also show 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. We are currently implementing a prototype of the proposed system. The objective is to better assess the proposed model and to demonstrate its applicability for wide deployment. Addressing the security issues of the architecture is part of our future work.

Acknowledgements

We are grateful to Stefan Saroiu of the University of Washington for sharing the Gnutella data with us. We would like to thank the editor and the anonymous reviewers for their valuable suggestions and detailed comments. We thank Ahsan Habib and Leszek Lilien for their valuable comments. This research is sponsored in part by the National Science Foundation grants ANI-0219110, CCR-0001788, and CCR-9895742.

References

[1] Available from Akamai home page <http://www.akamai.com>.
[2] S. Banerjee, B. Bhattacharjee, C. Kommareddy, G. Varghese, Scalable application layer multicast, in: Proceedings of ACM SIGCOMM'02, Pittsburgh, PA, USA, August 2002, pp. 205–220.
[3] P. Barford, J. Cai, J. Gast, Cache placement methods based on client demand clustering, in: Proceedings of IEEE INFOCOM'02, New York, June 2002.
[4] A. Bestavros, S. Mehrotra, DNS-based Internet client clustering and characterization, in: Proceedings of the 4th IEEE Workshop on Workload Characterization (WWC01), Austin, TX, USA, December 2001.
[5] K. Calvert, M. Doar, E. Zegura, Modeling Internet topology, IEEE Communications Magazine 35 (1997) 160–163.
[6] E. Choen, S. Shenker, Replication strategies in unstructured peer-to-peer networks, in: Proceedings of ACM SIGCOMM'02, Pittsburgh, PA, USA, August 2002, pp. 177–190.
[7] Y. Chu, S. Rao, S. Seshan, H. Zhang, A case for end system multicast, IEEE Journal on Selected Areas in Communications 20 (8) (2002) 1456–1471.
[8] D.E. Comer, Internetworking with TCP/IP: Principles, Protocols, and Architectures, fourth ed., Prentice-Hall, Englewood Cliffs, NJ, 2000.
[9] F. Dabek, M. Kaashoek, D. Karger, D. Morris, I. Stoica, H. Balakrishnan, Building peer-to-peer systems with chord, a distributed lookup service, in: Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII), Elmau/Oberbayern, Germany, May 2001, pp. 71–76.
[10] H. Deshpande, M. Bawa, H. Garcia-Molina, Streaming live media over peer-to-peer network, Technical Report, Stanford University, 2001.
[11] M. Hefeeda, A. Habib, B. Bhargava, Cost-profit analysis of a peer-to-peer media streaming architecture, CERIAS TR 2002-37, Purdue University, October 2002.
[12] K. Hua, Y. Cai, S. Sheu, Patching: A multicast technique for true video on-demand services, in: Proceedings of ACM Multimedia'98, Bristol, UK, September 1998.
[13] S. Jin, A. Bestavros, A. Iyengar, Accelerating Internet streaming media delivery using network-aware partial caching, in: Proceedings of IEEE ICDCS'02, Vienna, Austria, July 2002.
[14] B. Krishnamurthy, J. Wang, On network-aware clustering of web clients, in: Proceedings of ACM SIGCOMM'00, Stockholm, Sweden, August 2000.
[15] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, Oceanstore: an architecture for global-scale persistent storage, in: Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00), Boston, MA, USA, November 2000, pp. 190–201.
[16] A. Mahanti, D. Eager, M. Vernon, D. Sundaram-Stukel, Scalable on-demand media streaming with packet loss recovery, IEEE/ACM Transactions on Networking 11 (2) (2003) 195–209.
[17] Z. Miao, A. Ortega, Proxy caching for efficient video services over the Internet, in: Proceedings of 9th International Packet Video Workshop (PV'99), New York, USA, April 1999.
[18] T. Nguyen, A. Zakhor, Distributed video streaming over Internet, in: Proceedings of Multimedia Computing and Networking (MMCN02), San Jose, CA, USA, January 2002.
[19] S. Nilsson, G. Karlssom, Fast address lookup for Internet routers, in: Proceedings of Algorithms and Experiments (ALEX98), Trento, Italy, February 1998, pp. 9–18.
[20] The network simulator, Available from <http://www.isi.edu/nsnam/ns/>.
[21] University of Oregon Route Views Project, Available from <http://antc.uoregon.edu/route-views/>.
[22] V. Padmanabhan, K. Sripanidkulchai, The case for cooperative networking, in: Proceedings of 1st International

Workshop on Peer-to-Peer Systems (IPTPS '02), Cambridge, MA, USA, March 2002.
[23] V. Padmanabhan, H. Wang, P. Chou, K. Sripanidkulchai, Distributing streaming media content using cooperative networking, in: Proceedings of NOSSDAV'02, Miami Beach, FL, USA, May 2002.
[24] Available from Digital Island Home Page <http://www.digitalisland.com>.
[25] Available from Freenet Home Page <http://www.freenet.sourceforge.com>.
[26] Available from Gnutella Home Page <http://www.gnutella.com>.
[27] Available from Kazaa Home Page <http://www.kazaa.com>.
[28] Available from Multi-Threaded Routing Toolkit Home Page <http://www.mrtd.net/>.
[29] Available from Napster Home Page <http://www.napster.com>.
[30] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of ACM SIGCOMM'01, San Diego, CA, USA, August 2001.
[31] A. Rowstron, P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, in: Proceedings of 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany, November 2001.
[32] A. Rowstron, P. Druschel, Storage management in past, a large-scale, persistent peer-to-peer storage utility, in: Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01), Chateau Lake Louise, Banff, Canada, October 2001.
[33] M. Ruiz-Sanchez, E. Biersack, W. Dabbous, Survey and taxonomy of IP address lookup algorithms, IEEE Network 15 (2) (2001) 8–23.
[34] S. Saroiu, P. Gummadi, S. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of Multimedia Computing and Networking (MMCN02), San Jose, CA, USA, January 2002.
[35] S. Sen, L. Gao, J. Rexford, D. Towsley, Optimal patching schemes for efficient multimedia streaming, in: Proceedings of IEEE INFOCOM'99, New York, USA, March 1999.
[36] S. Sen, J. Rexford, D. Towsley, Proxy prefix caching for multimedia streams, in: Proceedings of IEEE INFOCOM'99, New York, USA, March 1999.
[37] I. Stoica, R. Morris, M. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for Internet applications, in: Proceedings of ACM SIGCOMM'01, San Diego, CA, USA, August 2001.
[38] D. Tran, K. Hua, T. Do, Zigzag: an efficient peer-to-peer scheme for media streaming, in: Proceedings of IEEE INFOCOM'03, San Francisco, CA, USA, April 2003.
[39] Y. Wang, Z. Zhang, D. Du, D. Su, A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers, in: Proceedings of IEEE INFOCOM'98, San Francisco, CA, USA, April 1998.
[40] D. Xu, H. Chai, C. Rosenberg, S. Kulkarni, Analysis of a hybrid architecture for cost-effective media distribution, in: Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN 2003), Santa Clara, CA, January 2003.
[41] D. Xu, M. Hefeeda, S. Hambrusch, B. Bhargava, On peer-to-peer media streaming, in: Proceedings of IEEE ICDCS'02, Vienna, Austria, July 2002.
[42] B. Yang, H. Garcia-Molina, Efficient search in peer-to-peer networks, in: Proceedings of IEEE ICDCS'02, Vienna, Austria, July 2002.
[43] B. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

Mohamed M. Hefeeda is a Ph.D. candidate with the Department of Computer Sciences at Purdue University, West Lafayette. He received the M.S. degree in computer science and engineering from University of Connecticut, Storrs in 2001, and the B.Sc. degree in Electronics Engineering from ElMansoura University, Egypt in 1994. His research interests include computer networks, peer-to-peer systems, multimedia networking, network economics, and network security. He is a student member of the IEEE Computer Society and the ACM SIGCOMM.

Bharat K. Bhargava is a professor of the Department of Computer Sciences and Department of Electrical & Computer Engineering at Purdue university since 1984. Professor Bhargava is conducting research in security issues in mobile and ad hoc networks. This involves host authentication and key management, secure routing and dealing with malicious hosts, adaptability to attacks, and experimental studies. Related research is in formalizing evidence, trust, and fraud. Applications in e-commerce and transportation security are being tested in a prototype system. He has proposed schemes to identify vulnerabilities in systems and networks, and assess threats to large organizations. He has developed techniques to avoid threats that can lead to operational failures. The research has direct impact on nuclear waste transport, bio-security, disaster management, and homeland security. These ideas and scientific principles are being applied to the building of peer-to-peer systems, cellular assisted mobile ad hoc networks, and to the monitoring of QoS-enabled network domains. He serves on six editorial boards of international journals (Transactions on Mobile Computing, Wireless Communications and Mobile Computing, International Journal of Computers and Applications, Multimedia Tools and Applications, International Journal of Cooperative Information Systems, Journal of System Integration).

His research group consists of nine Ph.D. students and two post doctors. He has six NSF funded projects. In addition, DARPA, IBM, Motorola, and CISCO are providing contracts and gift funds.

Professor Bhargava was the chairman of the IEEE Symposium on Reliable and Distributed Systems held at Purdue in October 1998. In the 1988 IEEE Data Engineering Conference, he and John Riedl received the best paper award for their work on "A Model for Adaptable Systems for Transaction Processing." Professor Bhargava is a Fellow of the Institute of Electrical and Electronics Engineers and of the Institute of Electronics and Telecommunication Engineers. He has been awarded the charter Gold Core Member distinction by the IEEE Computer Society for his distinguished service. He received Outstanding Instructor Awards from the Purdue chapter of the ACM in 1996 and 1998. In 1999 he received IEEE Technical Achievement award for a major impact of his decade long contributions to foundations of adaptability in communication and distributed systems. In 2003, he has been inducted in the Purdue's book of great teachers.

David K.Y. Yau received the B.Sc. (first class honors) degree from the Chinese University of Hong Kong, and the M.S. and Ph.D. degrees from the University of Texas at Austin, all in computer sciences. From 1989 to 1990, he was with the Systems and Technology group of Citibank, NA. He was the recipient of an IBM graduate fellowship, and is currently an Associate Professor of Computer Sciences at Purdue University, West Lafayette, IN. He received an NSF CAREER award in 1999, for research on network and operating system architectures and algorithms for quality of service provisioning. His other research interests are in network security, value-added services routers, and mobile wireless networking.