P2Cast: Peer-to-peer Patching for Video on Demand Service [1]

Yang Guo, Kyoungwon Suh, Jim Kurose, and Don Towsley
Department of Computer Science
University of Massachusetts, Amherst, MA 01003
yguo, kwsuh, kurose, towsley@cs.umass.edu

Abstract

Providing video on demand (VoD) service over the Internet in a scalable way is a challenging problem. In this paper, we propose P2Cast—an architecture that uses a peer-to-peer approach to cooperatively stream video using *patching* techniques, while only relying on unicast connections among peers. We address the following two key technical issues in P2Cast: (1) constructing an application overlay appropriate for streaming; and (2) providing continuous stream playback (without glitches) in the face of disruption from an early departing client. Our simulation experiments show that P2Cast can serve many more clients than traditional client-server unicast service, and that it generally out-performs multicast-based patching if clients can cache more than 10% of a stream's initial portion. We handle disruptions by delaying the start of playback and applying the shifted forwarding technique. The threshold in P2Cast, i.e, the length of time during which arriving clients form a single session, can serve as a "knob" to adjust the balance between the scalability and the clients' viewing quality.

I. Introduction

Providing video on demand (VoD) service over the Internet is a challenging problem. The difficulty is twofold. First, it is not an easy task to stream video on an end-to-end basis because of a video's high bandwidth requirement and long duration. Second, scalability issues arise when attempting to service a large number of clients. In particular, a popular video can attract a large number of viewers that issue requests asynchronously. Traditional VoD service employs a client-server unicast service model. Each client sets up its own connection with the server over a unicast channel. As the video popularity increases, the server needs to serve a large number of clients

[1]This research was supported in part by the National Science Foundation under NSF grants EIA-0080119, ITR-0085848, and ANI9977635. Any opinions, findings, and conclusions or recomendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

and soon becomes the bottleneck. How to optimally provide VOD service in a scalable manner remains an open question to be solved.

Several approaches have been explored in the past to tackle scalability issues faced by VoD service. IP multicast has been proposed to enhance the efficiency of one-to-many and many-to-many communication over the Internet. A series of IP Multicast-based schemes, such as Patching [1, 2], Periodic Broadcast [3, 4], and Stream Merging [5], have been developed that can drastically decrease the aggregate bandwidth requirement of the clients at the server by leveraging native IP multicast. For instance, under Patching, clients arriving close in time form a session. The server begins to multicast the entire stream at the client playback rate upon the arrival of the first client. The following clients retrieve the stream from the multicast channel and obtain the missing initial portion, denoted as the *patch*, from the server over a unicast channel. It has been shown that the required server bandwidth grows as the square root of the request arrival rate [2]. Unfortunately, IP multicast has not been widely deployed, making an IP multicast approach infeasible in practice.

Other approaches to addressing the scalability issue include proxy caching and the use of content distribution networks (CDNs). Here the content is pushed from the server to proxies or CDN servers close to the clients. By strategically placing a large number of servers around the Internet, clients can choose the server that incurs the least amount of congestion. Although this method can mitigate scalability concerns, it cannot resolve the issue entirely. For example, assume a proxy is assigned to serve the clients from a region. If the number of requesting clients is very large, this proxy can be overwhelmed.

Recently, peer-to-peer networks (P2P) have been used for file sharing, application-level multicast, and more [6–10]. Peer nodes bring computation and storage resources into the system, thus reducing the workload placed on the server and thereby increasing the overall scalability.

It is an intriguing technical question if these techniques can be integrated to tackle the scalability issue faced by the VoD service. In this paper, we propose P2Cast - an architecture based on a peer-to-peer approach to cooperatively stream video using the *patching* technique, while only relying on unicast connections among peers. In P2Cast, clients arriving close in time (within a *threshold*) form a session. For each session, the server, together with the P2Cast clients, form an application-level multicast tree over the unicast-only network. The entire video is streamed over the application-level multicast tree, so that it can be shared among clients. For clients who arrive later than the first client in the session and thus miss an initial segment of the video, the segment can be retrieved from the server or other clients that have already cached that initial segment. The clients in P2Cast are “active” in the sense that they can forward the video stream to other clients, and also cache and serve the initial portions of a video to other clients. Every P2Cast client actively contributes its bandwidth and storage space to the P2Cast system while taking advantage of the resources located at other clients. We compare the performance of P2Cast with IP multicast-based patching and the traditional unicast-based client-server approach. Simulation experiments show that P2Cast can serve many more clients than traditional client-server unicast service, and generally out-performs multicast-based patching if clients can cache more than 10% of stream’s initial part in our experiments.

Although P2Cast is based on patching, it is not a simple extension. The following issues need to be properly addressed before patching can be successfully applied.

- *Constructing the application overlay appropriate for streaming.* An application-level multicast tree having sufficient bandwidth to transmit the stream must be constructed and maintained. When a new client arrives, P2Cast needs to select a patch server that can serve the missing initial part of the video.
- *Providing continuous stream playback (without glitches) in the face of disruption from departing clients.* When clients leave the application overlay, P2Cast needs to overcome disruptions due to departures by restructuring the application overlay.

We develop the *Best Fit (BF) algorithm* to construct the application-level streaming delivery tree, as well as select the patch server that will serve the missing part of the video to an arriving client. We further investigate two variations of the BF algorithm, namely the BF-delay and the BF-delay-approx algorithm.

We handle disruptions by (1) delaying the start of playback; and (2) applying the shifted forwarding technique to the base stream. The threshold serves as a “knob” that can adjust the balance between the scalability and the clients’ viewing quality in P2Cast.

In summary, the major contributions of this paper are as follows:

- We propose P2Cast, a technique for providing VoD service, that scales better than a unicast-based client-server service approach and an IP multicast-based patching approach.
- We develop a series of overlay construction algorithms suitable for a video streaming service such as P2cast.

- We investigate techniques that provide continuous playback in the face of disruptions.

The remainder of the paper is organized as follows. In Section II we describe P2Cast. In Section III we present the Best Fit algorithm along with two variations. Section IV is dedicated to performance evaluation. Recovery from client departures and underlying network failures is investigated in Section V. Section VI overviews related work. Our conclusions and future work are included in Section VII.

II. P2CAST: PEER-TO-PEER PATCHING SCHEME

A. *Overview of P2Cast*

P2Cast is an architecture that uses a peer-to-peer approach to cooperatively stream video using *patching*, while only relying on unicast connections among peers. The key idea of P2Cast is to have each client act as a server while it receives the video. The non-scalability of traditional client-server unicast VoD service lies in the fact that the server is the only "contributor" to serving a video, and can thus become "swamped" by a large number of clients passively requesting the service. In the client-server service model, a client sets up a direct connection with the server to receive the video, and an amount of bandwidth equal to the playback rate is consumed along the route. As the number of requests increases, the bandwidth at the server and in the network is increases as well, and incoming requests must eventually be rejected.

In contrast, P2Cast clients not only receive the requested stream, but also contribute to the overall VoD service by forwarding the stream to other clients and caching and serving the initial part of the stream. Associated with P2Cast is a *threshold*, T. Clients that arrive within the threshold constitute a *session*. Together with the server, clients belonging to the same session form an application-level multicast tree, denoted as the *base tree*. The server streams the entire video over the base tree. We denote this complete video stream as the *base stream*. When a new client joins the session, it joins the base tree and begins receiving the ongoing base stream, which is being transmitted on the base tree. Meanwhile, the new client must obtain a "patch" containing the initial part of the video from the start of the session (when the video begins streaming on the base tree) to the time it joined the base tree. As we will see, the newly joining client obtains the patch from the server or another client. P2Cast clients behave like peers in a P2P network, and provide the following two functions:

- *Base Stream Forwarding*. P2Cast clients need to be able to forward the received base stream

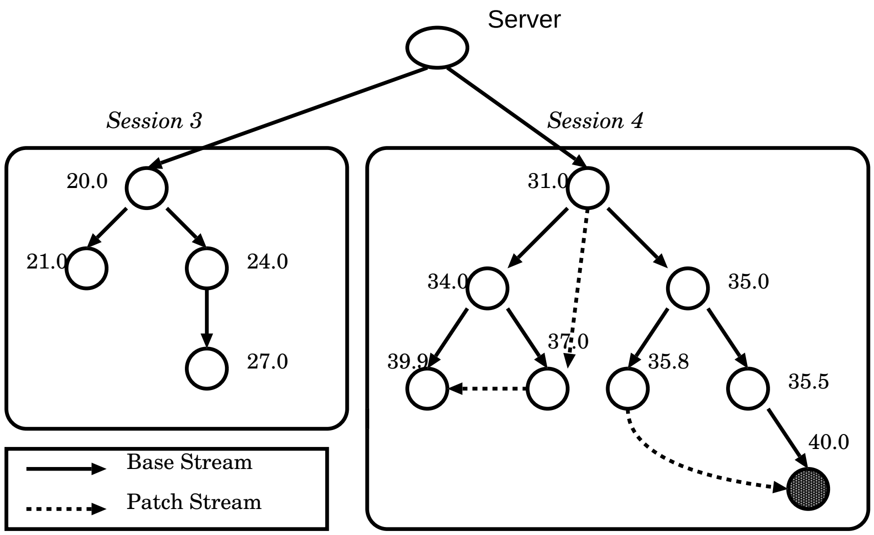


Fig. 1. A snapshot of P2Cast at time 40. Clients in a session form an application-level multicast tree together with the server. All clients in session 3 have finished patch retrieval; while 3 clients in session 4 are still receiving the patch stream from their parent patch servers.

to other clients so that clients and the server can form an application-level multicast tree over which the base stream is transmitted.

- *Patch Serving*. P2Cast clients need to have sufficient storage to cache the initial part of the video. A P2Cast client can then serve the patch to other clients.

We use an example to illustrate P2Cast. Fig. 1 illustrates a snapshot of P2Cast at time 40. It shows two sessions, session 3 and session 4, starting at time 20.0 and 31.0, respectively, with the threshold equal to 10. We use circles to represent clients with the client's arrival time marked beside the circle. A solid line with an arrow is used to represent the parent-child relationship in the base tree; and a dashed line with an arrow is used to represent the patch server-client relationship. The server and the clients in a session form an application-level multicast tree to deliver the base stream. At time 40, all clients in session 3 have finished patch retrieval, while three clients in session 4 are still in the process of receiving a patch stream. Note that clients belonging to different sessions are independent of each other. In Section V we will use this observation to improve the clients' robustness against disruptions caused by other clients' early departure.

We describe below a new client admission process, the base tree construction/joining process, the patch server selection process, and the failure recovery process, in turn.

B. New client admission

A new client first contacts the server. This allows the VoD service provider to keep track of the clients who have requested the service. The disadvantage of this approach is that the server becomes the single contact point. In practice, a VoD service provider can deploy multiple servers and use mapping techniques to guide clients to different servers to achieve the load balancing. Here we focus on the single server scenario.

As with the patching scheme proposed in the IP multicast setting [1, 2], all clients arriving within the threshold form a session. A new client that cannot join the most recent session starts a new session. The server streams the entire video to this client.

If the new client belongs to an already existing session, i.e., the difference between the first client's and the new client's arrival time is smaller than the threshold, it tries to join this session's base tree. In addition, the new client tries to select a patch server in its session that can stream the patch to it. If the new client successfully joins the base tree and selects a patch server, it is admitted. Because of limited bandwidth at the server and in the network, if a new client is not able to find a path with sufficient bandwidth to join the base tree, or to obtain the patch from a peer client or the server, the client will be rejected.

P2Cast combines the patch server selection process with the base tree joining process in order to help minimize a client's joining delay. The detailed algorithm used in P2Cast is illustrated in Section III.

C. Base tree construction

P2Cast employs the tree-first approach to construct the base tree. In general, there are two basic approaches for constructing an application-level multicast tree: the mesh-first approach and the tree-first approach. The mesh-first approach [6] builds up a mesh among the participating nodes first. The mesh is usually optimized for an application requirement and is dynamically adjusted to accommodate the underlying network change. For instance, if a new arrival or node departure/failure occurs, the mesh is restructured to adapt to the change. A routing algorithm is run at each node. In the tree-first approach [7, 8], the application-level multicast tree is created directly (without construction of the underlying mesh). The arrival of new nodes or departure/failure of existing nodes triggers the restructuring of the tree.

One design goal of P2Cast is to make the client as simple as possible. The mesh-first approach in [6] requires all participants to run a distributed algorithm to maintain the mesh, as well as a routing algorithm to route the traffic to the right peer nodes. In contrast, the nodes in the tree-first approach need only perform a simple data forwarding function. Moreover, in P2Cast, there are frequent arrivals of new clients, which will keep disturbing the mesh construction and thus affect the overall performance. Based on the above considerations, we choose the tree-first approach. Below we list the design principles followed in the base tree construction in P2Cast.

- *Bandwidth first principle*. VoD service has a stringent bandwidth requirement but is relatively insensitive to the delay. A "fat pipe" (i.e., a path with abundant unused bandwidth) is more likely to offer good quality and be robust to transient network congestion. Therefore we prefer to select a node with a fat pipe to the client.
- *Local information only principle*. Since the number of clients is large and dynamically varies over time, we want to avoid requiring that a node has global information, such as the number of clients in the tree, the structure of the tree, etc.. In the process of base tree construction, only local information should be used. By local information, we mean the information about this node itself, its parent node, and its child nodes.

For a new client, the base tree joining process starts with the server. Streaming media service requires a minimum amount of available bandwidth from a parent node to a child node. The server measures the available bandwidth from itself to the new client, and decides whether this client can be its child node. If the server admits the new client, this client joins the base tree and receives the base stream from the server. Otherwise, the server redirects the new client to one of its existing child clients, denoted as *candidate client*. The candidate client makes its own decision as to whether to admit this new client to be its child node. If not, the new client is further re-directed to its child node. The process continues recursively until the client successfully joins the base tree, or is rejected.

D. Patch server selection

A *patch server* serves the patch to a new client. Except for the first client in a session (who receives the entire video (base stream) from the server), all other clients will miss the initial part of the video and will require a *patch*. A new client needs to select a patch server from which it can obtain a unicast patch.

Since the server stores the entire video, it can always be a patch server as long as it has sufficient bandwidth. A peer client that arrives earlier and has sufficient bandwidth can also be a patch server. Fig. 2 illustrates an example. Assume that the session starts at time 0. At time t_1, client 1 arrives. It joins the base tree and receives the base stream starting at point t_1. We assume that client 1 obtains the patch stream from the server directly. At time t_2, client 2 arrives. Client 2 joins the base tree and receives the base stream from the base tree. Since client 1 has already cached part of the patch required by client 2 at time t_2 (the shaded part is the content already cached in client 1); and client 1 continues to receive its own patch and base stream, it can serve as the patch server for client 2.

The patch server selection process for a new client is identical to a new client's base tree joining process. In fact, all existing clients in a session arrive earlier than the new client, and thus can be its patch server if there is sufficient bandwidth. In Section V, however, we will consider patch recovery, where the arrival time of a candidate patch server needs to be compared with the arrival time of the requesting client.

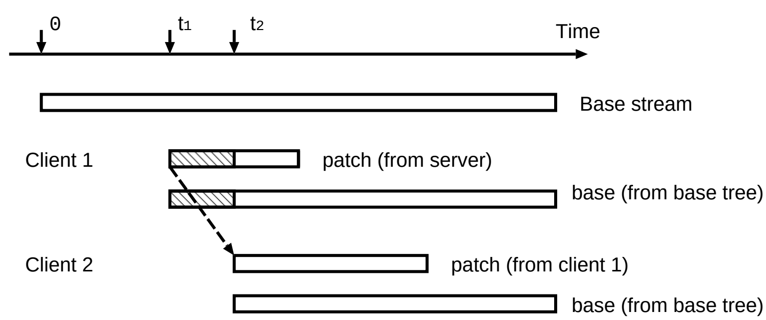


Fig. 2. The earlier arriving client (client 1) is eligible to be the patch server for the later arriving client (client 2) in P2Cast.

E. Failure Recovery

In P2Cast, the departure of a client or an underlying network failure, such as link or path failure or available bandwidth fluctuation, can disrupt both the delivery of a patch stream, as well as the transmission of the base stream over the application-level multicast tree. P2Cast provides two forms of failure recovery: *base stream recovery* and *patch recovery*.

We consider base stream recovery first. Suppose client A is disrupted by a failure. Due to the tree structure, all clients belonging to the subtree rooted at this client are affected by the failure. For the sake of simplicity and to prevent the server from receiving a large number of recovery requests, P2Cast only allows client A to contact the server and perform recovery. The recovery process is identical to that of a new client joining process except that here only the base stream is required. If the recovery attempt succeeds, the entire subtree is recovered. If it fails, client A is rejected. The children of client A will then contact the server and start the recovery process representing their own subtrees. This process continues recursively.

In patch recovery, assume client B seeks a new patch server. B contacts the server and begins the recursive recovery process identical to the server selection process for a new client. Note that only the clients arriving earlier than client B can be candidate patch servers. If a patch server is successfully selected, the recovery succeeds. Otherwise, client B is rejected, and the clients that receive a patch or the base stream from client B will initiate the recovery processes.

We present the signaling protocol used in P2Cast for clients to do the failure recovery in [11]. It consists of a network failure recovery protocol and a client departure recovery protocol, to deal with network failure and client departure, respectively. P2Cast clients constantly monitor the incoming traffic (both the base stream and patch stream) and when the incoming traffic's quality degrades to a certain degree, the recovery process is triggered.

In Section 4 we will further investigate how to provide continuous playback even in the face of disruptions . Below we first present the Best Fit algorithm for base tree construction and patch server selection.

III. Best Fit Algorithm for Base Tree Construction and patch server selection

In this section we first describe the *Best Fit (BF) algorithm* that constructs the base tree and selects the patch server in P2Cast. We then present two variations of BF algorithm, BF-delay and BF-delay-approx, both of which can also be used for base stream recovery and patch recovery.

A. Best-Fit (BF) Algorithm

In the Best Fit algorithm, the requesting client starts the recovery process by contacting the server. The following procedure is followed by the requesting client.

- Step 1. The requesting client N contacts a candidate parent P, starting with the server.
- Step 2. P estimates the bandwidth from P to N, $B(P, N)$. Meanwhile, it sends messages to all of its children in the base tree, denoted as $C(P)$, asking them to measure their respective

bandwidth to the requesting client.

• Step 3. P collects the measured bandwidth from its children, and identifies the child node C_{max} that has the fattest pipe to N, i.e., $C_{max} = argmax_{C \in C(P)}\{B(C,N)\}$. A tie is broken arbitrarily. There are two possibilities, depending on the measurement reported back to P: (a) Candidate node P has the fattest pipe to the requesting node N, $B(P,N) > B(C_{max}, N)$; and (b) one of the children has the fattest pipe to N, $B(P,N) \leq B(C_{max}, N)$. We discuss in turn each of these scenarios.

(1) If $B(P,N) > B(C_{max}, N)$, P has the fattest pipe to N and is able to support at least one stream. If N only requires the base stream, it can join the base tree using P as its parent node. If a patch is required, and P arrives earlier than N, then P becomes N's patch server. If both the base stream and a patch are required, the patch has the priority over the base stream. If P can serve the patch, it will become N's patch server. If P has sufficient leftover bandwidth to serve the base stream, N joins the base tree with P as parent node. If P cannot fully fulfill N's request, N is re-directed to C_{max}, and starts from the step 1 again.

(2) If $B(P,N) \leq B(C_{max}, N)$, then N is re-directed to C_{max}, and starts from step 1 above.

In step 3, if a client has out-degree constraint and cannot support any more clients, it can return an available bandwidth of zero to its parent client without conducting bandwidth measurement. If all candidate parents report zero bandwidth, the algorithm randomly selects one client to which N is re-directed.

B. BF-delay and BF-delay-approx algorithms

Here we further introduce two variations of BF: BF-delay and BF-delay-approx. In BF-delay, network delay information is used to break the tie at step 3, i.e., when multiple nodes have paths with the same amount of bandwidth to incoming client N, the client closest to the requesting client is selected.

BF-delay-approx uses a different bandwidth metric in steps 2 and 3. Instead of the actual available bandwidth $B(C,N)$, it uses $I(C,N)$, where $I(C,N)$ is 1 if client C has enough bandwidth to support the incoming client N's request, and 0 otherwise. The delay information is used to break the tie. Since BF-delay-approx only needs to test whether a client can admit the incoming client rather than measure the exact amount of available bandwidth (as requested by the BF and BF-delay algorithms), the measurement overhead of BF-delay-approx is lower than that of BF and BF-delay.

Hence the joining delay in BF-delay-approx is expected to be smaller.

IV. Performance Evaluation

In this section, we first evaluate the performance of P2Cast through simulation experiments. We then compare three overlay construction algorithms: BF, BF-delay, and BF-delay-approx. In our simulation results, the half-width of the 95% confidence interval of the data shown in this paper is always less than 5% of the point estimate. The experiments show that (1) P2Cast is more scalable than either a client-server unicast approach, or an IP multicast-based patching approach; (2) a larger threshold helps to serve more clients in P2Cast; and (3) under the same conditions, BF-delay and BF-delay-approx algorithm can serve more clients than BF and reduce the overall network workload over BF. However, they present a higher workload to the server than BF.

We will address the failure recovery problem in the Section V. For now we assume that no client departs early and that there are no network failures. We start with a description of the simulation setting.

A. Simulation setting

We use a 3-level network topology in our simulation experiment. Fig. 3 illustrates the top two levels generated by GT-ITM [12] with 100 nodes. We assume that each node is an abstraction of a local network that can host an unlimited number of clients, and that there is sufficient bandwidth within a local network to support media streaming. The network consists of one transit network (consisting of 4 nodes) and 12 stub domains. Shortest path routing is used to determine the routing.

We assume that the video playback rate is constant bit rate (CBR). We assign a bandwidth to each link in terms of the number of playback rates a link can support. The capacities of links between transit nodes and between transit nodes and stub domain nodes are chosen to be larger than those between stub domain nodes, since links in the network core are typically better provisioned and have more bandwidth than edge links. Using advanced coding techniques, videos with playback rates of 300Kbps to 500Kbps offer reasonably good viewing quality. A link with the capacity of 100Mbps can support 200 to 333 such streams. Since we simulate P2Cast providing service for one video, we choose the capacity of each core link to be 20, i.e., core links can support up to 20 streams simultaneously; and that of each edge link to be 5 for the simulation results reported in this paper. Furthermore, we change the location of the server from the transit network to the stub domain to study performance sensitivity to the server bandwidth change. We also vary the link capacity, and similar results are observed in [11].

We simulate the on-demand service of one video to clients whose arrival process is Poisson. Each client is equally likely to be placed at any node. It is possible for more than one client to reside at the same node.

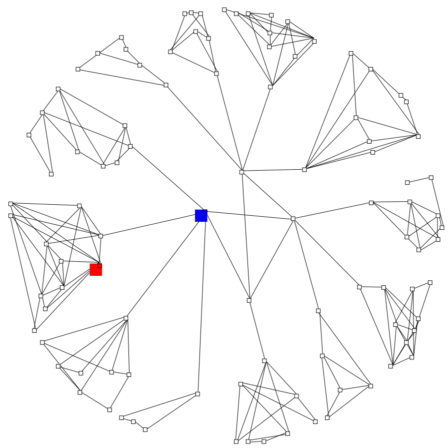


Fig. 3. Top 2-level network topology used in simulation

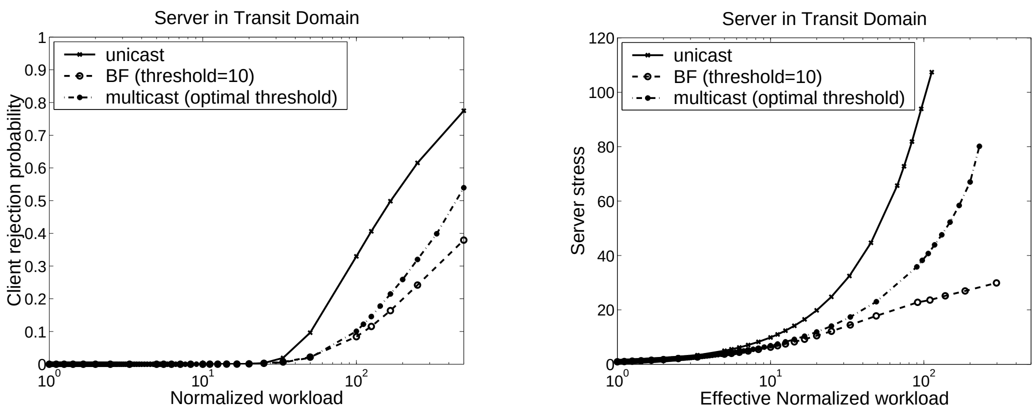
B. Notations and performance metrics

We denote by λ, L, and T as the average rate of clients arrival process (Poisson), the length of the video, and the threshold value, respectively. We define the *normalized workload (W)* to be the product of the average client arrival rate and the video length, $W = \lambda L$, and the *effective normalized workload* (W_e) to be the admitted normalized workload, $W_e = W(1-p)$. We use the following performance metrics in our evaluation: (1) *Rejection probability (p)* - the probability that a client cannot be admitted and served; (2) *Server stress (S)* - the average amount of bandwidth used at the server; (3) *Startup delay (d)* - the time that clients must wait before actually receiving the video and playing it back. In the following simulation we assume that the video length is 100 minutes.

C. Performance of P2Cast

In this section we present and discuss our simulation results.

• **Client rejection probability.** We first place the server in the transit domain (shaded node in the center of Fig. 3). We assume that the threshold of P2Cast is 10% of the video length, and every



(a) Rejection probability comparison

(b) Server stress comparison

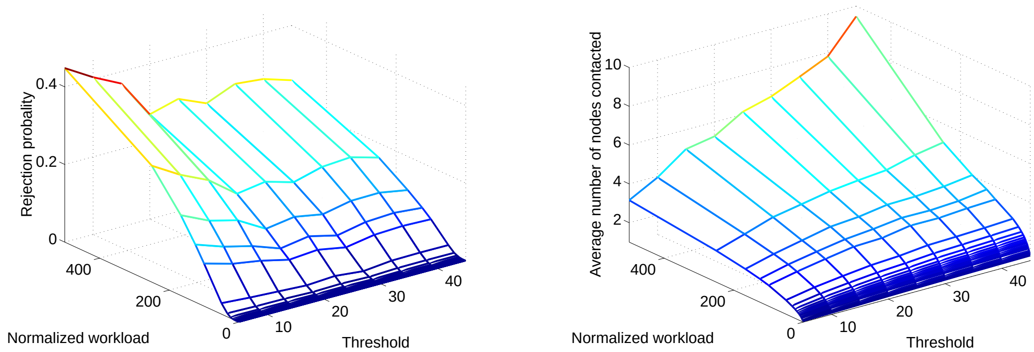
Fig. 4. Performance comparison of P2Cast, unicast, and IP multicast-based patching

client has sufficient storage space to cache a patch. As for IP multicast-based patching, we use the optimal threshold, which is $\min\{(\sqrt{2L\lambda+1}-1)/\lambda, L/2\}$ as derived in [2].

Fig. 4(a) depicts the rejection probability vs. the normalized workload for unicast, P2Cast using BF algorithm, and IP multicast-based patching. We observe that P2Cast admits more clients than unicast by a significant margin as the load increases. Also P2Cast outperforms the IP multicast-based patching, especially when the workload is high. These results show that the peer-to-peer paradigm employed in P2Cast helps to improve the scalability of P2Cast.

• **Server stress**. Fig. 4(b) shows the server stress for different schemes. The server is the most stressed under unicast. The server stress for P2Cast is even lower than that of IP multicast-based patching. Although application-level multicast is not as efficient as native IP multicast, the P2P paradigm employed in P2Cast can effectively alleviate the workload placed at the server - in P2Cast, clients take the responsibility off the server to serve a patch whenever possible.

• **Threshold impact on the performance of P2Cast.** In general the scalability of P2Cast improves as the threshold increases. Fig. 5(a) depicts the rejection probability of the P2Cast scheme with different thresholds. As the threshold increases, more clients can be admitted. Intuitively as the threshold increases, more clients arrive during a session and thus more clients can share the base stream. However, the requirement on the needed storage at each client also increases. We also observe in our experiment that the rejection probability decreases faster when the threshold is smaller than 20% of the video length, and flattens out afterwards. This suggests that we should



(a) Client rejection probability

(b) Startup delay

Fig. 5. Threshold impact on the performance of P2Cast

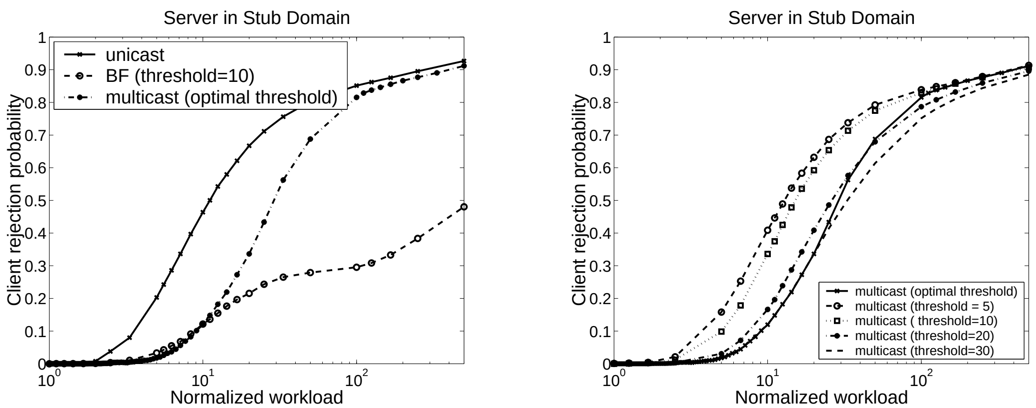
carefully choose the threshold such that most benefits can be obtained without overburdening the client.

In VoD service, the startup delay is another important metric. In the BF algorithm illustrated in Section III, every step incurs certain delay. However we expect that the available bandwidth measurement conducted at Step 2 is the most time consuming. Therefore the startup delay in P2Cast heavily depends on the number of candidate clients that a new client must contact before being admitted. We use the average number of candidate nodes a new client must contact before being admitted as an estimate of the startup delay.

Fig. 5(b) depicts the average number of nodes contacted for different threshold values in P2Cast. As either the normalized workload or the threshold value increases, the average number of nodes that need to be contacted also increases. Therefore although a larger threshold in P2Cast allows more clients to be served, it also leads to a larger joining delay.

• **Effect of server bandwidth.** We investigate the effect of server bandwidth on performance by moving the server to one of the stub nodes in Fig. 3 (a shaded node in the stub domain). Here the server has less bandwidth than when placed in the transit domain. Fig. 6(a) illustrates the client rejection probability. Overall, the rejection probability of all three approaches with the server in the stub domain is higher than that with the server in the transit domain (see Fig. 4(a)) due to the decreased server bandwidth.

Fig. 6(b) depicts the rejection probability of the IP multicast-based patching with the optimal



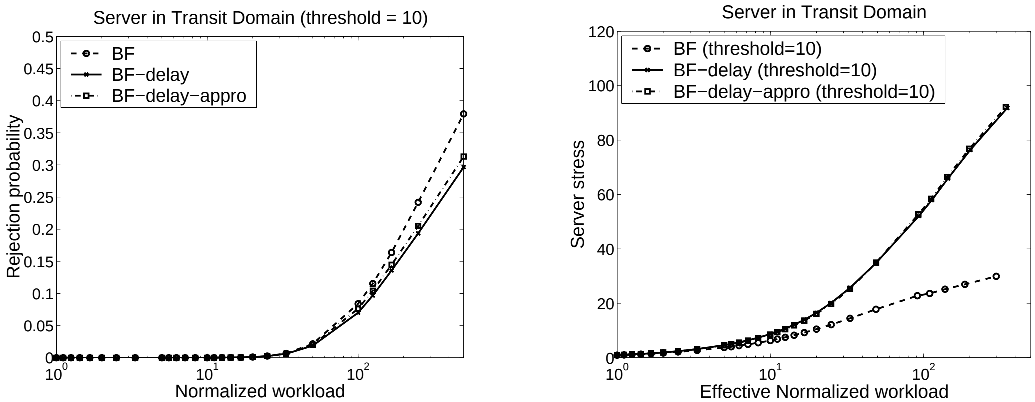
(a) Client rejection probability (b) Client rejection probability of IP multicast patching

Fig. 6. Effect of server bandwidth on performance with server placed in stub domain

threshold and several fixed thresholds. Interestingly, we observe that IP multicast-based patching with the optimal threshold performs badly when the normalized workload is high. The optimal threshold is derived to minimize the workload placed on the server, with the assumption that the server and the network have unlimited bandwidth to support the streaming service. As the client arrival rate increases, the optimal threshold decreases. This leads to an increasing number of sessions. Since one multicast channel is required for each session, with limited server bandwidth, the server cannot support a large number of sessions while providing patching service. This leads to the high rejection probability when the client arrival rate is high. Fig. 6(b) compares the IP multicast-based patching using the optimal threshold with that using the fixed thresholds of 5, 10, 20, 30 percent of the video size. When the arrival rate is high, the fixed larger threshold actually helps IP multicast patching to reduce the rejection rate. This suggests that "optimal threshold" may not be optimal in a real network setting.

D. Comparison of overlay construction algorithms

In Fig. 7, we compare the rejection probability, network usage, and server stress for BF, BF-delay, and BF-delay-approx, with the server placed in the transit domain. One observation is that both the BF-delay and BF-delay-approx outperform BF algorithm in terms of rejection probability. We also see that the performance of BF-delay and BF-delay-approx are close. Furthermore, we note that the server stress of BF is much less than that of BF-delay and BF-delay-approx. BF



(a) Client rejection probability (b) Server stress

Fig. 7. Comparison of the tree construction algorithms

encourages the requesting client to connect to a client with the most abundant bandwidth, even if that client is farther away from the requesting client than other candidate clients. Since bandwidth is consumed over more links, this potentially increases the rejection probability for future arrivals and the overall network usage. Nevertheless, by pushing requesting clients to other clients, the server is less stressed.

We also conducted experiments with the server in the stub domain, and similar results are observed [11].

V. FAILURE RECOVERY - PROVIDING CONTINUOUS PLAYBACK

In P2Cast, the departure of a client can disrupt both the delivery of a patch stream and the transmission of the base stream over the base tree. The ability to provide continuous playback in the face of these disruptions is essential to the success of P2Cast. In the following, we examine this issue from the following two aspects: (1) the disruption effect on the continuous playback of the patch stream; and (2) the disruption effect on continuous playback of the base stream. We find that the patch stream is relatively immune to the disruption because of its short duration. For the base stream, we proposed the shifted forwarding technique to conceal the disruption. Finally, the threshold in P2Cast serves as a knob that can adjust the balance between the scalability and the clients' viewing quality.

A. Disruption effect on the continuous playback of the patch

Let us consider the effect of a disruption on the patch first. In P2Cast, a client receives the patch from its patch server and begins play back immediately. The departure of the patch server interrupts this patch playback. However, the length of the patch is equal to the time difference of the clients' arrival time and the session starting time, which is no larger than the threshold and presumably much shorter than the video length. For instance, if the video length is 100 mins and the threshold is 10 mins, the patch length averages 5 mins in length, given the arrival process is Poisson. The short duration of the patch makes it relatively immune to disruption since the likelihood that the disruption hits the patch is much smaller than that of the base stream.

Fig. 8 plots the probability of the number of candidate clients contacted (i.e., the candidate patch servers contacted during the patch recovery, or the candidate base stream servers contacted during the base stream recovery) before a client can successfully recover from the failures. The left column of Fig. 8 is for the patch recovery; and the right column is for the base stream recovery. Since a client can be disrupted more than once, we collect the accumulated number of candidate clients contacted in recovery. The top two plots in Fig. 8 are for a threshold equal to 10% of the video length; and the bottom two plots are for a threshold equal to 30% of the video length. We assume that a client departs early with a probability of 0.1, and an early departing client is equally likely to depart at any point during the playback.

Most clients (0.996 for threshold 10% and 0.983 for threshold 30) will not be disrupted at all during the playback of a patch stream in our example. Patch delivery is disrupted only if the patch server departs while serving the patch. Since the length of patch is usually short, the chance that a patch gets disrupted is small.

Furthermore, the probability that a client has to contact more than 5 candidate clients during the patch recovery is less than 0.0003 and 0.0019, respectively, for the threshold of 10% and 30%. Thus if we can delay the playback for a short period and let the buffer build up a bit, say by delaying the playback to the time to contact 5 clients, the continuous playback of the patch stream can be provided with high probability.

B. Disruption effect on continuous playback of the base stream

The base stream is more prone to disruption because the entire subtree rooted at a departing client is disrupted. As depicted in Fig. 8, for the same environment, the probability that a client's

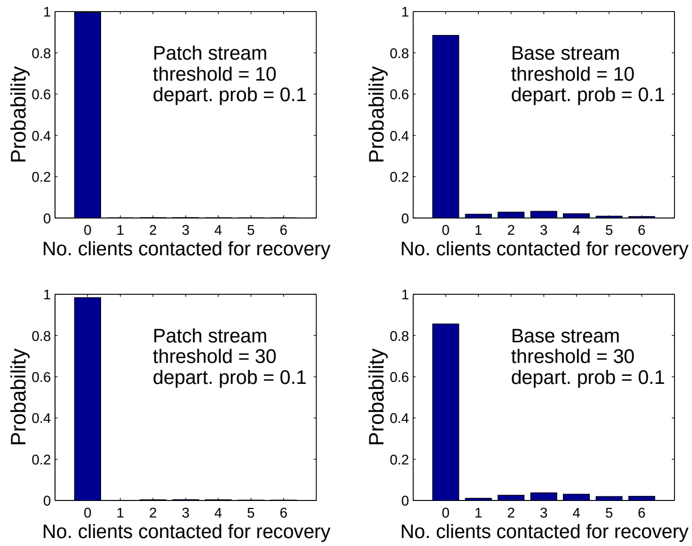


Fig. 8. Probability of no. candidate clients contacted for recovery caused by client departure.

base stream will not be disrupted goes down to 0.885 and 0.856, respectively. The probability that more than five clients need to be contacted increases to 0.0069 and 0.0205. However, note that the base stream is played back after being buffered for the period equal to the sum of playback delay and the patch length. We propose the *shifted forwarding technique* to protect the base stream from the glitch by using this cushion.

Shifted forwarding is similar to interval caching proposed for efficient memory caching. We give an example below to illustrate the idea. Suppose the session starts at time 0, and the client i's parent node departs at time t. Client i re-joins the base tree at time $t + \delta$. Thus client i and all its descendant clients in the base tree miss the video from time t to $t + \delta$. A glitch will be observed. Using shifted forwarding, client i will send a message to its parent client, asking it to forward the content starting at t (instead of forwarding the current data at $t + \delta$). Since the base stream has a cushion equal to the sum of the playback delay and patch size, the glitch can be avoided if the cushion is larger than δ. Usually the cushion (sum of playback delay and patch size, on the order of minutes) is much longer than the rejoining time (usually at the order of tens of seconds [13]), uninterrupted playback of the base stream can be achieved with high probability.

C. Threshold adjustment - balancing the scalability and viewing quality

The threshold value can be used as a knob to adjust the balance between scalability and viewing quality in P2Cast. As illustrated in Section IV, a larger threshold in P2Cast usually leads to the admission of more clients. However, a smaller threshold would help to provide better quality of the played out video - it is more likely that clients get continuous playback without a glitch.

A smaller threshold makes the patch size smaller, thus the patch disruption is less likely. Furthermore, a small threshold reduces the number of clients in a session. Hence the probability that the clients' base stream gets disrupted decreases. For instance, Fig. 9 depicts the probability that the base stream encounters at least one disruption during the playback if the base tree is a balanced binary tree. We assume that a node can leave with probability P_d, and a departure will affect all its descendant nodes. We note that the curve is concave and the probability decreases as the number of nodes decreases.

In the extreme, when the threshold is zero, the P2Cast reverts to the unicast service model, where clients' early departures do not affect each other and the continuous playback is guaranteed once a client is admitted. Therefore the threshold in P2Cast gives the service provider a knob to adjust the balance between the scalability and the clients' viewing quality.

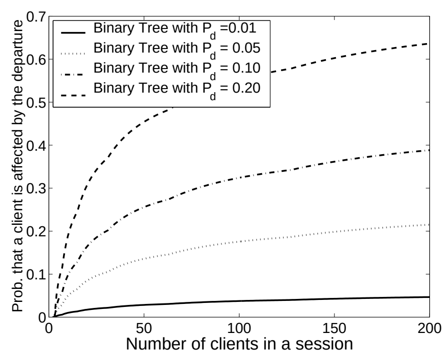


Fig. 9. Probability that the base stream encounters at least one disruption during the playback (balanced binary tree).

We also examine the probability that a client is forced to stop in the middle of video playback due to some other clients' early departure. This happens when a client cannot successfully recover from the disruption. For instance, a disrupted client cannot rejoin the base tree successfully. Fig. 10 depicts the probability of such forced early departure vs. clients' departure probability, P_d, with the threshold equal to 10% of the video length and the arrival rate set to be 1 arrival/min. Although this probability increases along P_d, overall the probability of forced early departure is small. Intuitively, although early departures disrupt other clients, their used bandwidth is released. Thus it is likely that disrupted clients can rejoin the base tree and find a new parent node.

Finally, we discuss how to use the threshold as a "knob" to balance the scalability with the

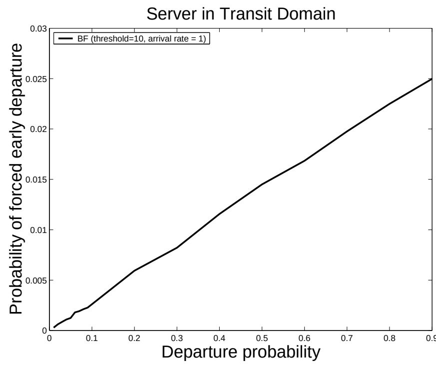


Fig. 10. Probability of forced early departure due to disruption.

viewing quality. As shown by the experiments presented in Section IV, the number of clients that can be serviced increases as the threshold increases. Meanwhile, the clients are more likely to experience the disruption during the course of watching a video as the threshold increases (see Fig. 9). The video streaming service providers can consciously use the threshold as the "knob" to adjust the balance between the scalability and the quality. In the following, we propose two techniques to tune the threshold.

- *Model-based threshold tuning.* Here we assume that the service providers can obtain both the scalability model and the viewing quality model. The scalability model reflects the relationship between the scalability and the length of threshold; while the viewing quality model reflects the relationship between the viewing quality and the length of threshold. For instance, the Fig. 5 can serve as the scalability model, and the Fig. 9 can serve as the quality model. Suppose that the desired client rejection probability is less than P_{scale}, and that the desired client disruption probability is less than $P_{quality}$. By monitoring client arrival rate, the service providers calculate the corresponding threshold T_{scale} and $T_{quality}$, respectively. The actual threshold, T, must be larger than T_{scale} while smaller than $T_{quality}$. There exit following two scenarios: (1) $T_{scale} \leq T_{quality}$. In this case, any $T \in [T_{scale} \;\; T_{quality}]$ would satisfy the requirement. (2) $T_{scale} > T_{quality}$. In this case, either scalability or view quality has to be sacrificed. The service providers will make the final decision.
- *Feedback-based threshold tuning.* The effectiveness of the model-based technique largely depends on the accuracy of the models. It is not a trivial task to obtain the accurate scalability and quality models in practice. In feedback-based threshold selection, we assume that the clients are willing and able to send quality-related feedbacks, such as the indication whether

they have been disrupted during the playback so far, back to the original server. The service providers monitor the feedbacks as well as the client arrival process, and periodically adjust the threshold to achieve the desired scalability-quality balance. For instance, the threshold can be adjusted using the following formula:

$$T_{n+1} = \begin{cases} T_n - \delta, & \text{Want better viewing quality} \\ T_n, & \text{Balanced scalability-quality} \\ T_n + \delta, & \text{Want more scalability} \end{cases} \tag{1}$$

The value of δ should be chosen significantly smaller than that of the threshold so that delicate scalability-quality balance can be achieved. The period at which the threshold is updated can be adjusted in practice until the best results to the service providers are achieved.

VI. Related Work

P2Cast is related to some previous works in the context of application level multicast systems, peer-to-peer steaming services, and available bandwidth measurement techniques.

To overcome the lack of IP multicast, many application-level multicast systems have been proposed recently, e.g., [6–9, 14–17]. Many of them use the delay as the single or primary metric in the tree construction, attempting to minimize the average delay among peer nodes. ZIGZAG[17] allows the media server to distribute content to many clients by organizing them into an appropriate tree rooted at the server. To satisfy the high bandwidth requirement of applications like streaming, the tree construction algorithms resort to limiting the out-degree of peer nodes. In contrast, P2Cast uses the information of measured available bandwidth in the base tree construction. The available bandwidth larger than the playback rate is guaranteed over the entire base tree.

Several peer-to-peer streaming systems have been developed [18–21]. Peercast [18] and CoopNet [19] exploit the cooperations from clients to distribute the content, thereby alleviating the load on the server. They are mainly designed for live media streaming. CoopNet also supports VoD service, and employs “distributed streaming” to obtain the content from multiple peers simultaneously. SplitStream [21] is a high-bandwidth content streaming/distribution system that is built upon Pastry [22], a generic substrate for peer-to-peer applications. There are several efforts put forth by industry, such as Allcast [23], vTrails [24], and Bluefalcon [25], that claim to provide live streaming and on-demand service. However, we cannot do a specific comparison because of the absence of published information on their on-demand services.

Finally, estimating the available bandwidth efficiently is very important for the overlay construction algorithm in P2Cast since it determines clients' joining delay and the likelihood of providing continuous playback in the face of disruptions. Reference [13] claims that their tool needs less than 15 seconds to produce an estimate of the available bandwidth. We expect the bandwidth measurement in P2Cast takes less time since the granularity of bandwidth of interest in P2Cast is the video playback rate. The measurement overhead can further be reduced if the BF-delay-approx algorithm is used. Furthermore, since the available bandwidth measurement has little impact to other traffic flows [13], we believe that the concurrent bandwidth measurement toward the same requesting client will not affect the measurement accuracy significantly.

VII. Discussion, Conclusions and Future Work

In this paper, we have described and evaluated a new technique, P2Cast, to tackle the scalability issue faced by VoD service over the Internet. P2Cast extends the IP multicast patching scheme to a unicast-only network by exploring the idea of P2P networking. We addressed two key technical issues in P2Cast, namely (1) constructing the application overlay appropriate for streaming; and (2) providing continuous stream playback (without glitches) in the face of disruption from an early departing client. Our simulations show that P2Cast scales much better than traditional client-server unicast service, and generally out-performs multicast-based patching if clients can cache more than 10% of stream's initial part. Furthermore, we handle disruptions by delaying the start of playback and applying the shifted forwarding technique to the base stream. The threshold in P2Cast serves as a knob that can adjust the balance between the scalability and the clients' viewing quality in P2Cast.

Future research can proceed in several directions. First, we are developing the middleware implementing P2Cast. How to design the system architecture of such a middleware, and how to enhance the current available streaming protocol to support P2cast, are both challenging and intriguing research questions. Second, as in any peer-to-peer networks, clients can behave selfishly by not contributing their resources and serving other peers. Furthermore, clients may be forced not to fully participate in the peer-to-peer service because of the practical limitations. For instance, the up-link bandwidth for end users that use ADSL or cable modem may not be large enough due to connections' asymmetric nature. Or the client behind the firewall may not be able to stream the video to the outside clients. We are studying mechanisms to overcome these limitations, and to encourage clients to cooperate with each other. We hope that a form of fairness among peers can be achieved in the future. Third, in this paper we point out that the threshold can serve as a knob to adjust the balance between the scalability and the clients' viewing quality in P2Cast. A technique for selecting the threshold based on the video's popularity and network conditions is worth further study. Finally, current P2Cast has been designed for supporting CBR videos. Extending this work to support VBR is also an interesting question for future research.

REFERENCES

[1] K. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proc. ACM Multimedia*, September 1998.

[2] L. Gao and D. Towsley, "Threshold-based multicast for continuous media delivery," in *IEEE Transactions on Multimedia*, December 2001.

[3] A. Hu, "Video-on-demand broadcasting protocols: A comprehensive study," in *Proc. IEEE INFOCOM*, April 2001.

[4] Y. Guo, L. Gao, D. Towsley, and S. Sen, "Seamless workload adaptive broadcast," in *Proc. of International Packetvideo Workshop*, April 2002.

[5] D. Eager, M. Vernon, and J. Zahorjan, "Bandwidth skimming: A technique for cost-effective video-on-demand," in *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, January 2000.

[6] Y. Chu, S. G.Rao, and H. Zhang, "A case for end system multicast," in *Proc. ACM SIGMETRICS*, June 2000.

[7] P. Francis, Y. Pryadkin, P. Radoslavov, R. Govindan, and B. Lindell, "Yoid: Your own internet distribution." 2001.

[8] L. Mathy, R. Canonico, and D. Hutchison, "An overlay tree building control protocol," in *Proc. International Workshop on Networked Group Communication*, November 2001.

[9] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An application level multicast infrastructure," in *Proc. USENIX Symp. on Internet Technologies and Systems*, March 2001.

[10] Kazaa, "http://www.kazaa.com,"

[11] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P^2cast: P2p patching scheme for vod service," tech. rep., Department of Computer Science, University of Massachusetts Amherst, http://www-net.cs.umass.edu/ yguo/p2castTech.ps, 2002.

[12] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOM*, April 1996.

[13] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput," in *Proc. ACM SIGCOMM*, August 2002.

[14] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole, "Overcast:reliable multicasting with an overlay network," in *Proc. USENIX OSDI Symp.*, October 2000.

[15] Y. Chu, S. G.Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *Proc. ACM SIGCOMM*, August 2001.

[16] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. ACM SIGCOMM*, August 2002.

[17] D. A. Tran, K. A. Hua, and T. T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *Proc. IEEE INFOCOM*, March 2003.

[18] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over peers," Tech. Rep. 2002-21, Stanford University, March 2002.

[19] V. Padmanabhan, H. Wang, P. Chou, and K. Sripandidkulchai, "Distributing streaming media content using cooperative networking," in *Proc. IEEE Workshop on NOSSDAV*, May 2002.

[20] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," in *Proc. IEEE ICDCS*, July 2002.

[21] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in a cooperative environment," in *Proc. IPTPS'03*, February 2003.

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

[23] Allcast, "http://www.allcast.com,"

[24] vTrails, "http://www.vtrails.com,"

[25] Bluefalcon, "http://www.bluefalcon.com,"