P^2Cast: P2P Patching Scheme for VoD 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

Computer Science Technical Report 02-34

July 10, 2002

Abstract

Providing video on demand (VoD) service over the Internet in a scalable way is a challenging problem. In this paper, we propose P^2Cast - 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 P^2Cast: (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 P^2Cast 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. A threshold on the length of time during which arriving clients are served in a single session in P^2Cast serves as a knob to adjust the balance between the scalability and the clients' viewing quality in P^2Cast.

I. Introduction

Providing video on demand (VoD) service over the Internet is a challenging problem. The difficulty is twofold. First it is not a trivial 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 soon becomes the bottleneck. The question of how to best provide VoD service to a large number of clients in a scalable manner remains 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 at the server by leveraging the native IP multicast. For instance, under Patching, clients arriving close in time form a session. The server begins

[1]This research was supported in part by the National Science Foundation under NSF grants EIA-0080119, NSF ANI-9973092, ANI9977635, ANI-9977555, and ANI9875513. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

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, and access to it is very limited so far, which makes the above schemes less applicable.

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 alleviate the scalability issue, 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, hence reducing the workload placed on the server and thereby increasing the overall scalability.

Whether these techniques can be coupled together to tackle the scalability issue faced by the VoD service is an intriguing technical question. In this paper, we propose P^2Cast - an architecture based on a peer-to-peer approach to cooperatively stream video using *patching* technique, while only relying on unicast connections among peers. In P^2Cast, clients arriving close in time (within the *threshold*) form a session. For each session, the server, together with the P^2Cast 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 that arrive later than the first client in the session and thus miss the initial part of the video, the missing part can be retrieved from the server or other clients that have already cached the initial part. The clients in P^2Cast are "active" in the sense that they can forward the video stream to other clients, and also cache and serve the initial part of video to other clients. Every P^2Cast client actively contributes its bandwidth and storage space to the P^2Cast system while taking advantage of the resources from other clients. We will show that the P^2Cast scales much better than either a unicast-based client-server service approach, or an IP multicast-based patching approach.

Although P^2Cast 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. In addition, when a new client arrives, P^2Cast needs to select a patch server that can serve the missing initial part of video.
- *Providing continuous stream playback (without glitches) in the face of disruption from departing clients*. When clients leave the application overlay, P^2Cast needs to overcome disruptions due to early 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 to serve the missing part of video. We further investigate two variations of BF algorithm, namely BF-delay and BF-delay-approx algorithm. We compare the performance of P^2Cast with IP multicast-based patching and the traditional unicast-based client-server approach. Simulation experiments show that P^2Cast can serve many more clients than traditional client-server unicast service, and generally out-performs the multicast-based patching if clients can cache more than 10% of stream's initial part in our experiments.

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

In summary, the major contributions of this paper lie in the following three aspects:

- Propose P^2Cast that scales better than a unicast-based client-server service approach and an IP multicast-based patching approach in providing VoD service.
- Develop a series of overlay construction algorithms suitable for video streaming service.
- Investigate the techniques to provide continuous playback in the face of disruptions.

The remainder of the paper is organized as follows. In Section II we describe P^2Cast. In Section III we present the Best Fit algorithm along with two variations. Section IV is dedicated to performance evaluation. Recovery from client early departure and underlying network failure is investigated in Section V. Section VI introduces related works, and conclusions and future work are included in Section VII.

II. P^2CAST: P2P PATCHING SCHEME

A. *Overview of P^2Cast*

P^2Cast 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 P^2Cast is to have each client act as a server while it receives the video. The un-scalability of traditional client-server unicast VoD service lies in the fact that server is the only "contributor", 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. 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 consumed and incoming requests must eventually be rejected.

In contrast, P^2Cast 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 P^2Cast is a *threshold*, T. The 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 retrieves the base stream from it. Meanwhile, the new client must obtain a "patch" - the initial part of the video from the start of the session to the time it joined the base tree. As we will see, it obtains the patch from the server or another client. P^2Cast clients behave like peers in a P2P network, and provide the following two functions:

- *Base Stream Forwarding*. P^2Cast clients need to be able to forward the received base stream to other clients so that clients and server can form an application-level multicast tree over which the base stream is transmitted.

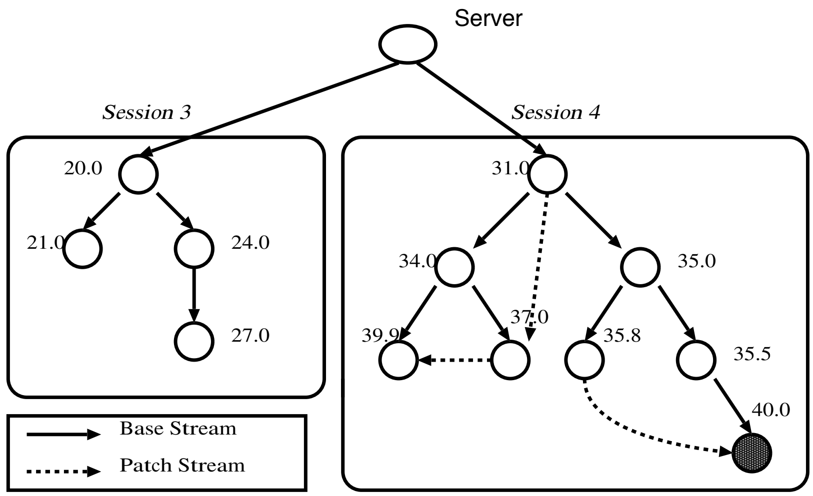


Fig. 1. A snapshot of P^2Cast at time 40. Clients in a session form an application-level multicast tree together with the server. All clients in session 3 have finished patch retrieval; while 3 clients in session 4 are still receiving the patch stream from their parent patch servers.

- *Patch Serving*. P^2Cast clients need to have sufficient storage to cache the initial part of the video. A P^2Cast client can then serve the patch to other clients.

We use an example to illustrate P^2Cast. Fig. 1 illustrates a snapshot of P^2Cast at time 40. It shows two sessions, session 3 and session 4, starting at time 20.0 and 31.0, respectively, with threshold equal to 10. We use circles to represent clients with a client's arrival time marked beside the circle. A solid line with an arrow is used to represent 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 the patch retrieval; while three clients in session 4 are still in the process of receiving the 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 the disruption caused by the clients' early departure.

Base stream forwarding and patch caching and serving should not pose a challenge to clients' machines given modern computers' computational power and storage space. As ADSL and cable modems become increasingly widely deployed, together with the advance of video coding technique, end users will have sufficient down-link bandwidth to receive two streams simultaneously (one for base stream and one for patch stream). Currently, the up-link bandwidth for some end users that use ADSL or cable modem may not be large enough because of connections' asymmetric nature. These clients can receive P^2Cast service even though they cannot contribute by serving other clients. We expect there to be a number of clients that reside at places with good network connections; and the bandwidth of ADSL and cable modem will continue to improve. In the following discussion, we assume that clients have symmetric network connection unless stated otherwise.

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

B. New client admission

A new client first contacts the server. This allows the VoD service provider to keep track of the clients that 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. Meanwhile, 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 the 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.

P^2Cast combines the patch server selection process with the base tree joining process in order to help minimize the clients' joining delay. The detailed algorithm used in P^2Cast is illustrated in Section III.

C. Base tree construction

P^2Cast employs the tree-first approach to construct the base tree. In general, there are two types of approaches in constructing the 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 toward the application requirement and is dynamically adjusted to accommodate 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. The arrival of new nodes or departure/failure of existing nodes triggers the restructure of the tree.

One design goal of P^2Cast 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 only need to provide simple data forwarding function. Moreover, in P^2Cast, 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 P^2Cast.

- *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 would like to select the node with the fat pipe to the client.
- *Local information only principle*. Since the number of clients is large and clients can come and go, we want to avoid a requirement that any 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 that 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 that can unicast the patch to it.

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 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, 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.

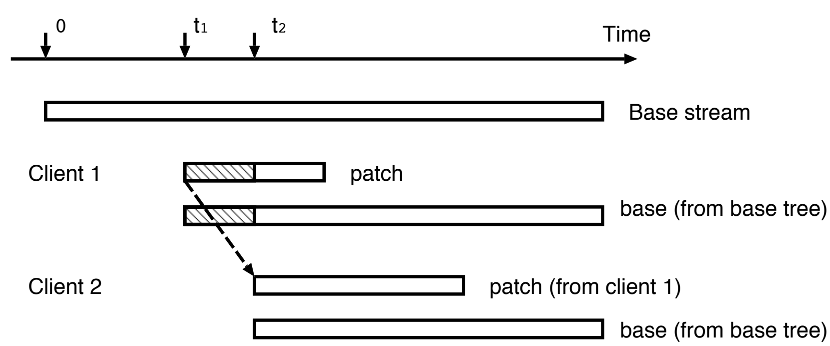


Fig. 2. The earlier arriving client (client 1) is eligible to be the patch server of the later arriving client (client 2) in P^2Cast.

E. Failure Recovery

P^2Cast requires a failure recovery mechanism to overcome the disruptions caused by departing clients. The departure of a client can disrupt both the delivery of a patch stream from a patch server, as well as the transmission of the base stream over the application-level multicast tree. P^2Cast provides two types of failure recovery: *base stream recovery* and *patch recovery*.

We consider the 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, P^2Cast 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. It contacts the server and begins the recursive recovery process identical to the server selection process for a new client. Note that here only the clients that arrived earlier than client B can be candidate patch servers. If a patch server is successfully selected, the recovery succeeds. Otherwise, client B has to be rejected, and the clients that receive a patch or the base stream from client B will initiate the recovery processes.

In Section 4 we will further investigate how to provide continuous playback even in the face of disruptions from clients' early departure and network failure. We show that by delaying the start of playback and employing the *shifted forwarding technique*, we can alleviate the impact of disruption. Furthermore the threshold serves as a knob that can adjust the balance between the scalability and the clients' viewing quality in P^2Cast. Below we present the signaling protocols used in P^2Cast for clients to do failure recovery. Although a network failure has the same impact as a client departure, it has to be handled differently in recovery. We present the network failure recovery protocol and the client departure recovery protocol to deal with the network failure and client departure, respectively.

E.1 Network failure recovery protocol

P^2Cast clients detect the network failure by constantly monitoring the incoming traffic, both the base stream and patch stream. If the incoming traffic's quality is below certain threshold, the client sends a TEST_ERROR message to its corresponding parent node and wait for the response. For example, if base stream's quality is bad, the message is sent to the parent node in the base tree. If patch stream has problem, the message is sent to the patch server. At the same time a timer is started. There are three possible scenarios.

- *No response.* If no response arrives before the timer expires, another TEST_ERROR message is sent and the timer is re-started. If no response received after 3 trials, the client determines that the network connection between its parent node and itself is down. It contacts the server and initiates the failure recovery process.
- *NETWORK_RECOVER message received.* In this case although the parent node can receive TEST_ERROR message, the parent node infers that the available bandwidth between itself and child node is not sufficient to

support the streaming anymore. For instance, the parent node receives the stream from its parent node with good quality and forwards the stream to the child client. If child client still complains the deteriorated quality, it suggests that the available bandwidth between parent node and child node is not sufficient. The parent will send NETWORK_RECOVER message to child client and encourage the child client to initiate the recover process.

- *WAIT message received.* The parent client asks the client to wait while it is contacting its own parent client, or is in the recovery process. The client starts a timer, and will resent TEST_ERROR message if the problem does not go away before timer expires.

E.2 Client early departure recovery protocol

Once a client decides to leave, it immediately sends DEPART_RECOVER message to all of its child nodes in the base tree, as well as the clients that are receiving patches from it. Upon receiving DEPART_RECOVER message, the client realizes that its immediate parent node or patch server is going to leave, the client contacts server and initiates the recovery process. If the recovering clients receive TEST_ERROR message from their descendant clients, they simply send back WAIT message. If a client suddenly crashes, the protocol used for network failure recovery can be used to handle this situation.

So far we have described P^2Cast service in this section. In the following section we will present the Best Fit algorithm for construction of the base tree and the 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 P^2Cast. We then present two variations of BF algorithm, BF-delay and BF-delay-approx algorithms. All these algorithms can also be used for the base stream recovery and patch recovery.

A. *Best-Fit (BF) Algorithm*

In the Best Fit algorithm, the requesting client starts the 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. Depending on the measurement reported back to P, there follow two scenarios: (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 the patch is required, and P arrives earlier than N, then P becomes N's patch server. If both the base stream and the 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.

B. BF-delay algorithm and BF-delay-approx algorithm

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 to the incoming client N with the same amount of bandwidth, the client closest to the requesting client is selected.

BF-delay-approx uses the different bandwidth metric in step 2 and step 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 BF and BF-delay algorithm do, 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 small.

IV. Performance Evaluation

In this section, we first evaluate the performance of P^2Cast through simulation experiments. We then compare three overlay construction algorithms - BF algorithm, BF-delay algorithm, and BF-delay-approx algorithm. 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) P^2Cast is more scalable than either a client-server unicast approach, or an IP multicast-based patching approach; (2) the larger threshold helps to serve more clients in P^2Cast; 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 there are no network failures. We start with the introduction 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 [11] with 100 nodes. We assume that each node is an abstraction of a local network that can host an unlimited number of clients, and 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. The shortest path algorithm 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 rate 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 core network are typically better provisioned and have more bandwidth than the edge links. 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. However, we do change the location of server from the transit network to the stub domain to study the sensitivity of the performance to the server bandwidth change. We also vary the link capacity to investigate its impact.

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.

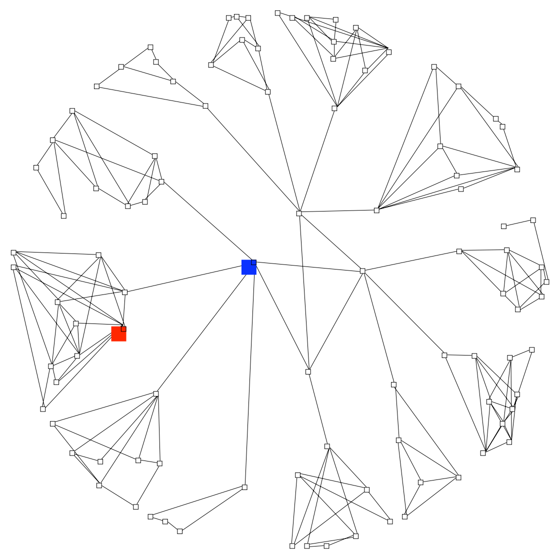


Fig. 3. Top 2-level network topology used in simulation

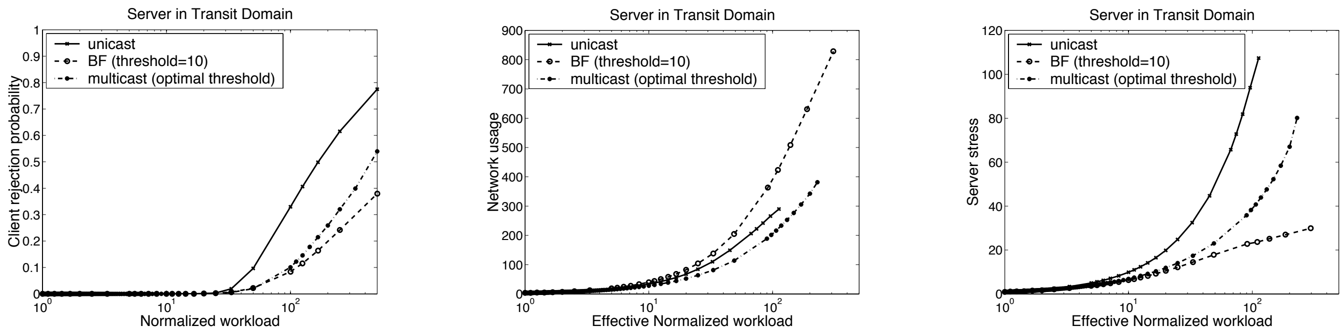
B. Notations and performance metrics

We denote by λ, L, and T the average arrival rate of clients, the length of the video, and the threshold value, respectively. We use the following performance metrics in our evaluation: (1) *Rejection probability (p)* - the probability that a client cannot be admitted and served; (2) *Normalized workload (W)* - the product of average client arrival rate and video length, $W = \lambda L$; (3) *Effective normalized workload (W_e)* - the admitted normalized workload, $W_e = W(1-p)$; (4) *Average workload posed on the network/Network usage (U)* - the sum of all link usage, $U = \sum_i u_i$, where the link usage of link i, u_i, is the product of link utilization and its capacity. Hence network usage represents the average workload placed on a network; (5) *Server stress (S)* - the average amount of bandwidth used at the server; (6) *Startup delay (d)* - the time that clients have to 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 P^2Cast

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 P^2Cast is 10% of the video length, and every client has sufficient storage space to cache the patch. As for IP multicast-based patching, we use the optimal threshold, which is $\min\{(\sqrt{2L\lambda+1} -$



(a) Rejection probability comparison (b) Workload comparison (c) Server stress comparison

Fig. 4. Performance comparison of P^2Cast, unicast, and IP multicast-based patching

$1)/\lambda, L/2\}$ as derived in [2].

Fig. 4(a) depicts the rejection probability vs. the normalized workload for unicast, P^2Cast using BF algorithm, and IP multicast-based patching. We observe that P^2Cast admits more clients than unicast by a significant margin as the load increases. Also P^2Cast outperforms the IP multicast-based patching, especially when the workload is high. It shows that the peer-to-peer paradigm employed in P^2Cast helps to improve the scalability of P^2Cast.

• **Average workload posed on network (network usage).** Since some clients are rejected by the server due to bandwidth constraints, we use *effective normalized workload* to represent the actual workload presented by the clients.

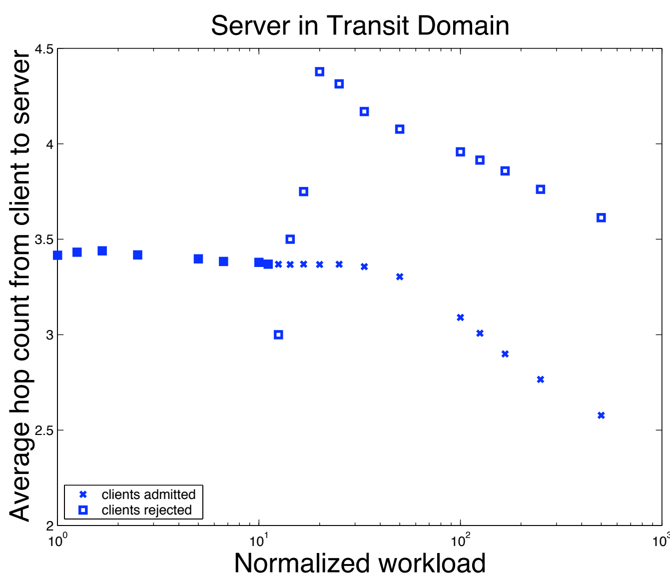
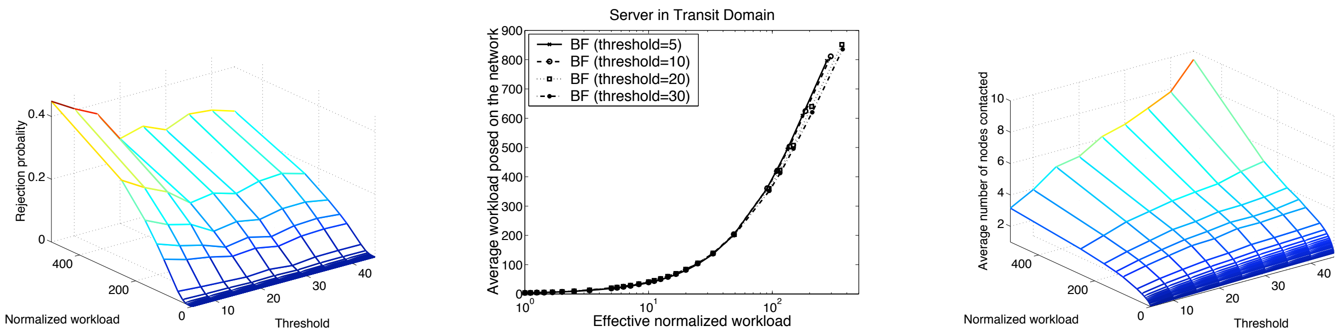


Fig. 5. Average hop count from server to clients in unicast service

Fig. 4(b) illustrates the network usage vs. the effective normalized workload. Since application-level multicast is not as efficient as native IP multicast, P^2Cast poses more workload on the network than IP multicast-based patching. Interestingly, we find that P^2Cast has higher network usage than the unicast service approach. In Fig. 5 we plot the average hop count from the admitted clients to the server for both P^2Cast and unicast. The average hop count



(a) Client rejection probability (b) Average workload posed on the network (c) Startup delay

Fig. 6. Threshold impact on the performance of P^2Cast

in unicast tends to be much smaller than that in P^2Cast, suggesting that a unicast-based service selectively admits clients that are closer to the server. Intuitively, clients closer to the server are more likely to have sufficient bandwidth along the path. This may explain why the network usage of a unicast service is lower than that of P^2Cast.

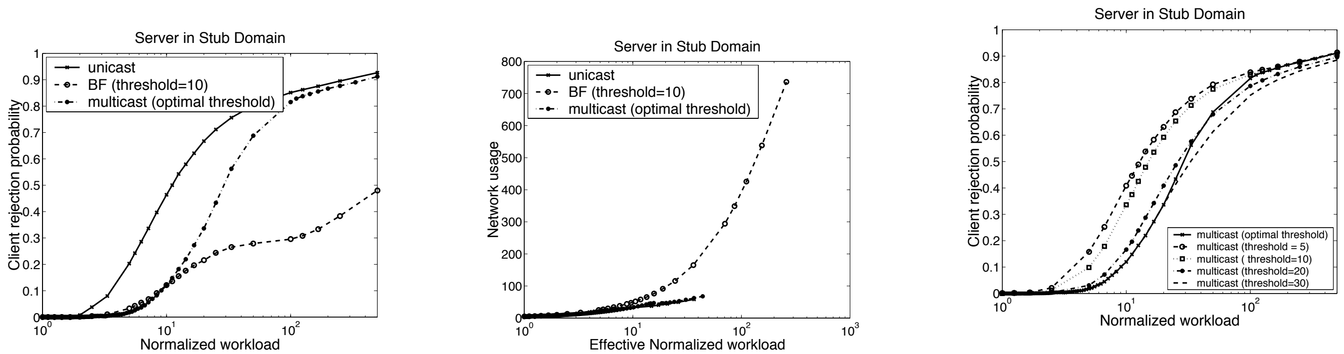
• **Server stress**. Fig. 4(c) shows the server stress for different schemes. The unicast server is most stressed. The server stress for P^2Cast is even lower than that of IP multicast-based patching. Although the application-level multicast is not as efficient as native IP multicast, the P2P paradigm employed in P^2Cast can effectively alleviate the workload placed at the server - in P^2Cast, clients take the responsibility off the server to serve the patch whenever possible.

• **Threshold impact on the performance of P^2Cast.** In general the scalability of P^2Cast improves as the threshold increases. Fig. 6(a) depicts the rejection probability of P^2Cast 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 storage space posed on 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 carefully choose the threshold such that most benefits can be obtained without overburdening the client.

Fig. 6(b) shows that the difference in the average workload imposed on the network is marginal as a function of threshold in P^2Cast.

In VoD service, startup delay is another important metrics. The startup delay is the time that clients have to wait before actually receiving the video and playing it back. In our simulation, we use the average number of nodes a client has to contact before being admitted to estimate the startup delay. Fig. 6(c) depicts the average number of nodes needs to be contacted for different threshold in P^2Cast. As the normalized workload increases, the number of nodes in a session increases and the height of the base tree is more likely to be large. Hence the average number of nodes needs to be contacted also increases. For different threshold, as threshold becomes larger, more clients in a session, thus larger average number of contacted nodes. It suggests that although larger threshold in P^2Cast helps to admit more clients, it also leads to larger playback delay.

• **Effect of server and network bandwidth.** We investigate the effect of server bandwidth on the performance by



(a) Client rejection probability

(b) Average workload posed on the network

(c) Client rejection probability of IP multicast patching

Fig. 7. Effect of server bandwidth on performance with server placed in stub domain

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 being placed in the transit domain. Fig. 7(a) and Fig. 7(b) illustrate the rejection probability and average workload placed on the network. Overall, the rejection probability of all three approaches with server at stub domain is higher than that with server in transit domain (see Fig. 4(a)) due to the decreased server bandwidth. As to the average workload on network, P^2Cast again generates the most workload. Similarly, the admitted clients in unicast tend to be closer to the server and this trend is even more evident than with server in transit domain (see Fig. 8).

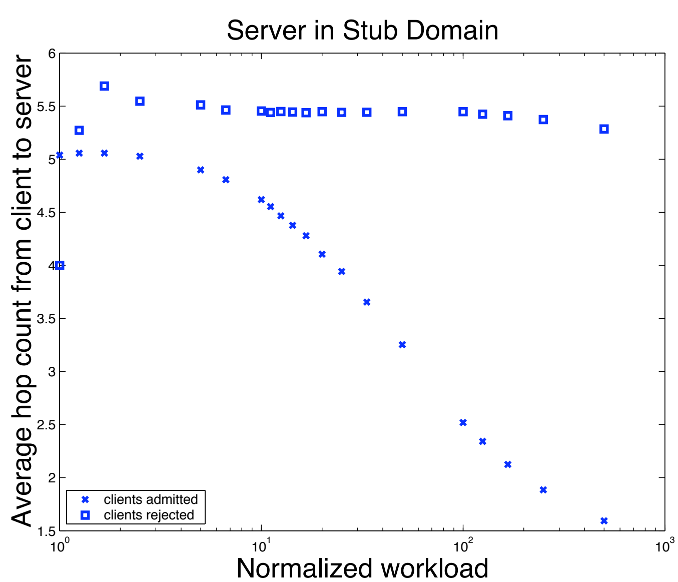
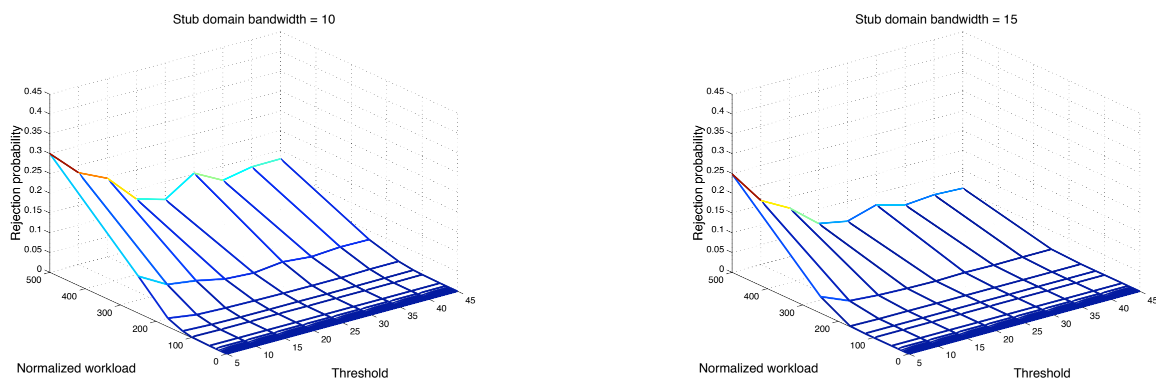


Fig. 8. Average hop count from server to clients vs. normalized workload

Fig. 7(c) depicts the rejection probability of IP multicast-based patching with the optimal threshold and several fixed thresholds. Interestingly, we observe that the IP multicast-based patching with optimal threshold performs badly when the normalized workload is high. The optimal threshold is derived to minimize the workload posed on the server, with the assumption that the server has infinite bandwidth. As the client arrival rate increases, the optimal



(a) Edge link bandwidth = 10

(b) Edge link bandwidth = 15

Fig. 9. Impact of stub domain bandwidth to the client rejection probability

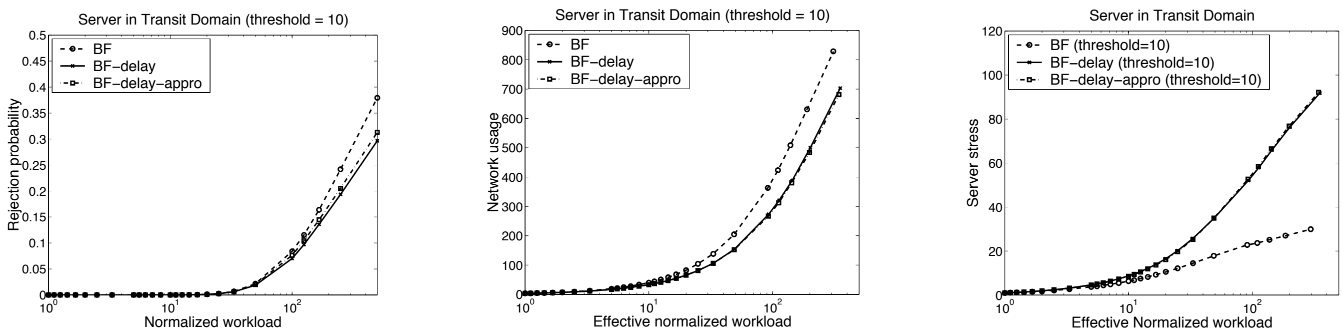
threshold decreases. This leads to an increasing number of sessions. Since one multicast channel is required for each session, with limited server bandwidth, the server cannot support a large number of sessions while providing patch service. This leads to the high rejection probability when the client arrival rate is high. Fig. 7(c) 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 the IP multicast patching to reduce the rejection rate. This suggests that "optimal threshold" may not be optimal in a real network setting.

We also vary the bandwidth with the server placed in the transit domain. Fig. 9(a) and 9(b) depicts the rejection probability with different threshold and workload for edge link bandwidth of 10 and 15, respectively. As the edge link bandwidth increases, the server can support more clients and thus rejection probability decreases. However both graphs exhibit similar trend: as the threshold increases, the rejection probability decreases quickly, and then flat out around the threshold of 20% of the video length.

D. Comparison of overlay construction algorithms

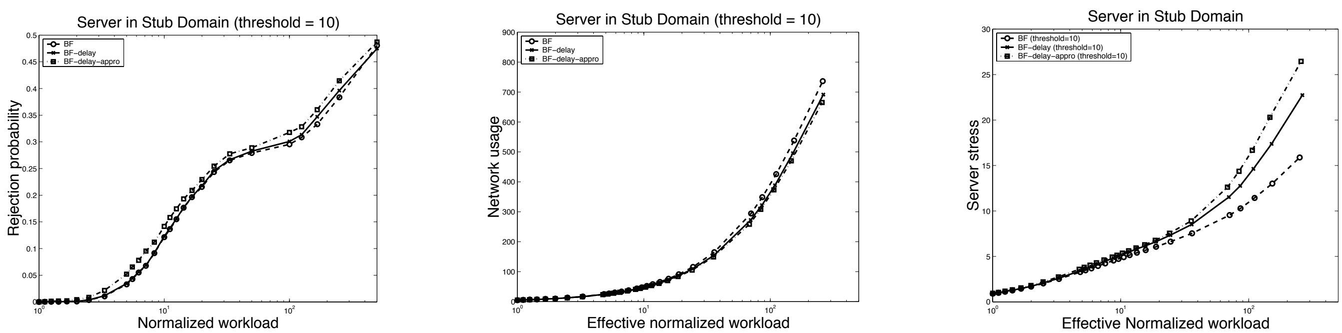
In Fig. 10, we compare the rejection probability, network usage, and server stress for BF, BF-delay, and BF-delay-approx, with the server placed at the transit domain. One observation is that both the BF-delay and BF-delay-approx outperform BF algorithm in terms of rejection probability and network usage. We also see that the performances 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 encourages the requesting client to connect to a client with 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 then conduct the experiments with the server in the stub domain (see Fig. 11). Similarly, BF-delay and BF-delay-appro algorithm outperforms BF in terms of rejection probability and network usage, and server stress is lower



(a) Client rejection probability (b) Average workload posed on the network (c) Server stress

Fig. 10. Comparison of the tree construction algorithms



(a) Client rejection probability (b) Average workload posed on the network (c) Server stress

Fig. 11. Comparison of the tree construction algorithms with server in stub domain

for BF algorithm. The differences of these performance metrics are not as significant as in the case with server in transit domain though, since the server is at the edge of the network and has less bandwidth.

V. Failure Recovery - Providing continuous playback

In P^2Cast, 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 P^2Cast. 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 P^2Cast 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 P^2Cast, 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 base stream.

Fig. 12 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. 12 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. 12 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 probability 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 in 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.

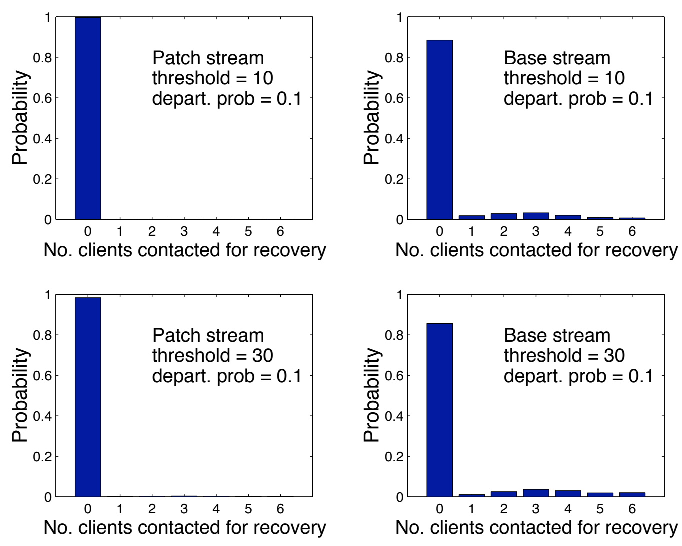


Fig. 12. Probability of no. candidate clients contacted for recovery caused by client departure

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. 12, for the same environment, the probability that a client's 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 seconds [12]), uninterrupted playback of 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 P^2Cast. As illustrated in Section IV, a larger threshold in P^2Cast 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 clients' base stream gets disrupted decreases. For instance, Fig. 13 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 P^2Cast 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 P^2Cast gives the service provider a knob to adjust the balance between the scalability and clients' viewing quality.

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. 14 depicts the probability of such forced early departure vs. clients' departure probability, P_d, with the threshold equal to 10% of 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.

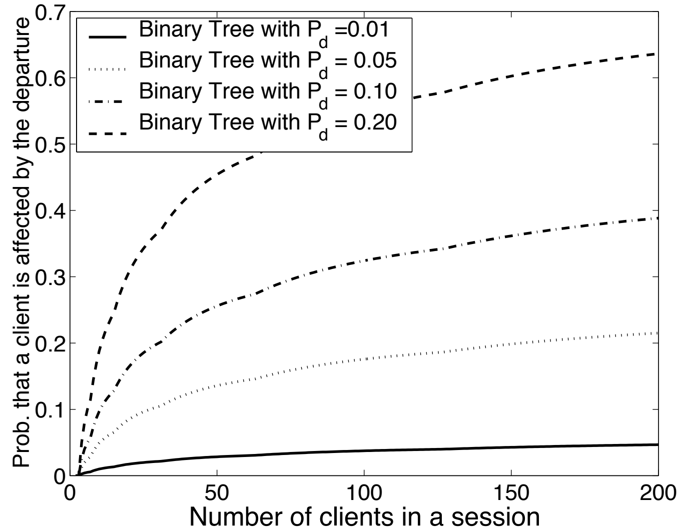


Fig. 13. Probability that the base stream encounters at least one disruption during the playback (balanced binary tree)

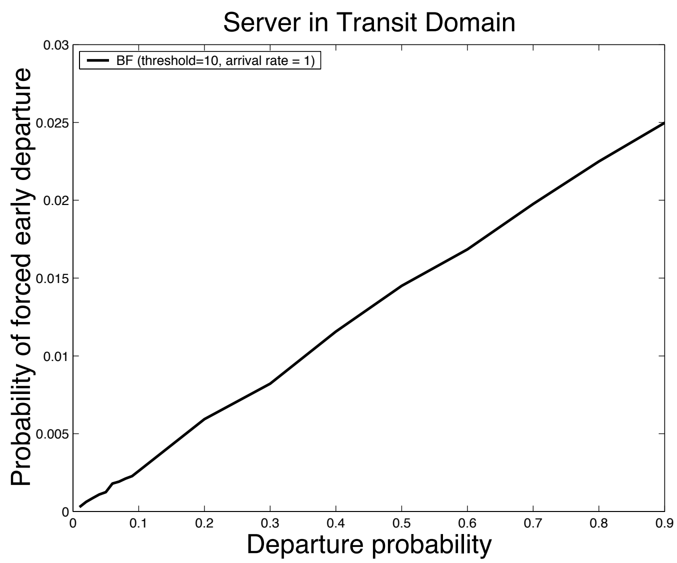


Fig. 14. Probability of forced early departure due to disruption

VI. Related Work

P^2Cast is related to some previous works in the context of IP multicast-based VoD schemes, overlay tree construction, P2P assisted teleconferencing and streaming service, and available bandwidth measurement.

Several application-level multicast tree construction algorithms have been proposed in the past. Narada [6], Yoid [7], TBCP [8], and ALMI [9] use the delay as the single or primary metric in the tree construction. ALMI and ESM [13] are designed to support small group size teleconferencing. ALMI constructs a minimum-spanning tree assuming that the global knowledge of clients and networks is available, while ESM is mainly based on the Narada work. Overcast [14] is the closest to P^2Cast in that it constructs an overlay tree trying to maximize the bandwidth. However, Overcast is proposed to provide a content delivery infrastructure on top of pre-deployed proxies.

P2P file sharing services such as Kazaa, Gnutella, and Napster share the spirit with our work in the sense that peer nodes collaborate with each other by serving cached contents to other peer node. However, P2P file sharing services do not provide VoD service. Peercast [15] and CoopNet [16] are designed for live media streaming. CoopNet also supports VoD service however it treats the video as a large file. There are several efforts put forth by industry, such as Allcast [17], vTrails [18], and Bluefalcon [19], 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 overlay construction algorithm in P^2Cast since it determines clients' joining delay and the likelihood of providing continuous playback in the face of disruptions. Reference [12] claims that their tool needs less than 15 seconds to produce an estimate of the available bandwidth. We expect the bandwidth measurement in P^2Cast takes less time since the granularity of bandwidth of interest in P^2Cast is the video playback rate. The measurement overhead can further be reduced if BF-delay-approx algorithm is used.

VII. Discussion, Conclusions and Future Work

In this paper, we propose P^2Cast to tackle the scalability issue faced by VoD service over the Internet. We propose P^2Cast that extends IP multicast patching scheme to the unicast-only network by exploring the idea of P2P network. We further address two key technical issues in P^2Cast, 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 simulation experiments show that P^2Cast scales much better than traditional client-server unicast service, and generally out-performs the 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 base stream. The threshold in P^2Cast serves as a knob that can adjust the balance between the scalability and the clients' viewing quality in P^2Cast.

Future research can proceed along several avenues. First, we plan to develop a middle-ware implementing P^2Cast. To see how it works over real networks will be very interesting. Second, 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 P^2Cast. How to actually select the threshold based on the video's popularity and network condition is worth further study. Third, current P^2Cast is designed with supporting CBR videos in mind. How to extend it to support VBR is an interesting question. Finally, we are considering other techniques to protect the patch from disruption other than delaying clients' playback, such as using advanced coding techniques or multiple backup paths.

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, D. Towsley, and J. Kurose, "Efficient schemes for broadcasting popular videos," in *Proc. Inter. Workshop on Network and Operating System Support for Digital Audio and Video*, July 1998.

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

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

[13] 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.

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

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

[17] Allcast, “http://www.allcast.com,”

[18] vTrails, “http://www.vtrails.com,”

[19] Bluefalcon, “http://www.bluefalcon.com,”