DirectStream: A Directory-based Peer-to-Peer Video Streaming Service

Yang Guo
Corporate Research
Thomson Inc.
Princeton, NJ 08540
Yang.Guo@thomson.net
Phone: 609-987-7725
Fax: 609-987-7299

Kyoungwon Suh, Jim Kurose, and Don Towsley
Department of Computer Science
University of Massachusetts at Amherst
Amherst, MA 01003, USA
kurose,towsley,kwsuh@cs.umass.edu
Phone: 413-545-1585
Fax: 413-545-1249

CMPSCI TR 07-30

Abstract— **Providing video streaming service over the Internet is a challenging task. Recent works [1–7] show that peer-to-peer networking is a promising technique to address the scalability issues faced by video streaming service. Peer-to-peer video streaming allows users distributed across the Internet to collaborate and bring computation and storage resources into the system, hence reducing the workload placed on the server and increasing the number of clients served. In this paper, we propose DirectStream, a directory based peer-to-peer video streaming service that efficiently and cost-effectively provides video on-demand service with VCR operation support. In addition, we develop an application-level multicast based directory service tailored for DirectStream, known as AMDirectory service. We analytically and experimentally examine the system performance, and show that the proposed scheme significantly reduces the workload posed on the origin server, and that it scales extremely well as the popularity of the video increases even if participating clients behave non-cooperatively. Furthermore, AMDirectory service allows DirectStream to perform well in the face of a large number of concurrent users. We also discuss how DirectStream supports basic VCR operations, and provides continuous playback in the presence of the users' early departures and bandwidth starvation.**

I. Introduction

Providing video streaming service in a scalable way is a challenging task due to the long durations and high bandwidth requirements of video streams. In the past, IP multicast based approaches[8–14] and replicated server/proxy based approaches[15–18] have been developed to tackle the scalability issue. However, all these methods have their own drawbacks, due to limited deployment and access to IP multicast, or to costs associated with deployment and maintenance. A video streaming service that is scalable and flexible while incurring low deployment and maintenance cost is desirable.

Recent works, e.g., [1–7], suggest that peer-to-peer networking is a promising technique to address the scalability issues faced by video streaming services. Peer-to-peer video streaming allows users distributed across the Internet to bring computation and storage resources into the system, hence reducing the workload placed on the video's origin server, and thereby admitting more users to receive the service. In this paper, we propose *DirectStream* for peer-to-peer video streaming with the support of a distributed directory service, known as *AMDirectory service*. DirectStream provides instantaneous or near-instantaneous video on-demand service (VoD), and enables basic VCR functionalities.

In DirectStream, a user is implemented with both server and client capabilities, and behaves like a peer—it caches a moving window of the latest received video content, and serves latecomers by continuously forwarding the cached content. A set of clients arrived close in time forms a forwarding tree through which a single stream from the server is shared. We study the system performance through both analysis and simulation, and show that the proposed scheme significantly reduces the workload posed on the server, and scales extremely well as the popularity of the video increases. We further investigate several key technical issues emerging from this peer-to-peer streaming service, namely

- *Combating non-cooperative users*. As in any peer-to-peer system, there is no guarantee that the peers will behave cooperatively. For DirectStream, selfish

users may always try to connect to the original server directly, or refuse to forward streams to other peers.

- *Constructing the peer-to-peer overlay appropriate for streaming*. The goal here is to allow users to have good quality streaming service, while maximizing the number of users that can be served by the system in the long run.
- *Providing scalable directory service that facilitates the overlay construction in the face of a large number of users*. As the video popularity increases, the number of users in the system also increases. How to provide directory service under such environments is a challenging task.

Our analysis and simulation experiments show that DirectStream performs well even when a fraction of the participating clients behave selfishly or non-cooperatively. We further offer the intuition why DirectStream is immune to such a challenge. As to the second issue, we employ a QoS-sensitive peer-selection algorithm to construct the appropriate streaming overlay. Given the available bandwidth between the candidate parents and the newly arrived user satisfies the bandwidth requirement, we choose the parent peer that give us better chance to admit more users in the long run. We develop AMDirectory service to provide scalable directory service tailored for DirectStream. AMDirectory service is an application-level multicast based directory service that dynamically distributes and tracks the location of users in the network. Since it offers user proximity information, an incoming user can quickly identify the appropriate parent user without searching through the entire candidate list, which may contain hundreds of candidates when the video popularity is high.

The remainder of the paper is organized as follows. Related work is included in Section II. In Section III, we describe the design and architecture of DirectStream. In Section IV, we present AMDirectory service. The performance of DirectStream is evaluated both analytically and through simulation in Section VI. In Section V, we discuss the means to support basic VCR operations, and to provide continuous playback in the presence of clients' early departures and bandwidth starvation. Finally, Section VII concludes the paper with summary and future directions.

II. Related work

DirectStream is related to previous works in the context of peer-to-peer video streaming, application-level multicast, and available bandwidth measurement techniques.

There have been significant efforts to address the scalability issue presented in the streaming media service using peer-to-peer networking techniques, e.g., [1–7]. The work reported in [1, 2, 5, 7] are mainly designed for live media streaming. SplitStream [5] is a high-bandwidth content streaming/distribution system that is built upon Pastry [19] and Scribe [20]. It provides live streaming service while DirectStream targets VoD service. AMDirectory service is also built upon Pastry and Scribe. However, AMDirectory is designed to provide the directory service for DirectStream, while the streaming overlay is constructed using peer selection. Distinct overlays help to suite individual overlays to their respective functionalities. Buffering a moving window of latest data at client side corresponds to the interval caching technique that was first proposed in [21, 22] to efficiently utilize memory for video streaming. It has been applied at the application level in several occasions ([23–28]). Hua *et al*. [23] first applied the client side interval caching in an ATM networking environment. Jin *et al*. [25] analyzed the link cost of a client-based caching approach. Sharma *et al*. [27] propose a distributed prefetching protocol that allows clients to prefetch and store portions of the streaming data ahead of play-out time so as to facilitate the recovery process caused by parent client' early departure. Cui *et al*. [26] analytically examined the server stress and link stress under different stream access pattern. This paper is an extension of [4]. While above studies focus on different aspects of applying buffering technique to p2p VoD streaming, to our best knowledge, we are the first to propose a framework that allows clients to take full advantage of the benefits of interval caching, and carefully examine the system's performance and the effect of clients' non-cooperative behavior unique to the peer-to-peer networks. Furthermore, we propose AMDirectory service that allows DirectStream to perform well in the face of large number of users, and investigate how to appropriately construct the peer-to-peer networks in order to improve the system's scalability.

AMDirectory service is also related to the work on IP and application-level anycast, e.g., [29–33]. The anycast service proposed in [33] also uses Pastry and Scribe. We designed AMDirectory to provide directory service for DirectStream, and tailor the design to work with peer-to-peer video streaming service.

Finally, estimating the available bandwidth efficiently is very important for the overlay construction since it determines clients' joining delay. Reference [34] claims that their tool needs less than 15 seconds to produce an estimate of the available bandwidth. We expect the bandwidth measurement in DirectStream to take less time since the granularity of bandwidth of interest in DirectStream is the video playback rate. Furthermore, since the available bandwidth measurement has little impact to other traffic flows [34], we believe that the concurrent bandwidth measurement toward the same requesting client will not affect the measurement accuracy significantly.

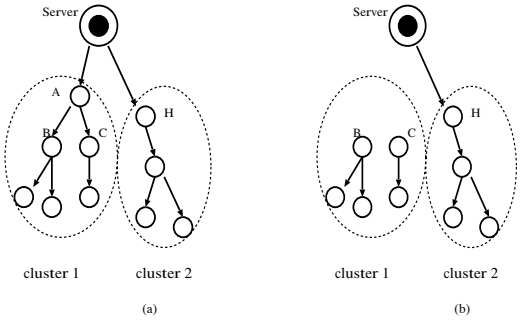


Fig. 1. DirectStream system. (**a**) DirectStream system with two clusters — one headed by client A and the other headed by client H. (**b**) DirectStream system after the departure of client A.

III. DIRECTSTREAM: A DIRECTORY-BASED PEER-TO-PEER VIDEO STREAMING SERVICE

DirectStream is comprised of clients, content servers, and the AMDirectory service. The content servers provide the same functionality as in the traditional client-server service model—storing contents in their repository and serving clients' requests so long as sufficient bandwidth is available. In contrast, the clients in DirectStream act as both a traditional client and as a server. Clients cache a moving window of video, and can serve other clients by forwarding the stream. A peer-to-peer overlay is formed among clients over which the video stream is forwarded. The AMDirectory provides lookup service, which keeps track of all servers and clients participating in DirectStream, and helps new clients to obtain the required service. The detailed design of AMDirectory service is presented in Section IV.

Fig. 1(a) illustrates a DirectStream peer-to-peer streaming system at time t with one server and a number of clients. These clients form two *clusters*, cluster 1 and cluster 2. We define cluster to be a set of clients that share a single stream from the server. A peer-to-peer streaming overlay is established among the clients of the same cluster. For instance, clients A, B, C and other three clients form the first cluster. The streaming overlay, in this case a forwarding tree, is established among them.

A cluster in DirectStream evolves over time. For instance, client A was a member of the cluster 1 however it left the cluster after completing playback and forwarding the stream to B and C (see Fig. 1(b)). This cluster may grow in the future by admitting new clients into it.

Below we first describe the data caching at clients and the usage of AMDirectory service in DirectStream. We then introduce a peer selection algorithm to construct the overlay appropriate for streaming. Finally, we present the procedure of servicing a new client.

A. Data caching at clients

Clients in DirectStream cache a moving window of the most recent content that they have received. Assume a client buffers b minutes worth of video, and is watching the video at a position τ minutes from the beginning of the video at time t. The client caches the most recent b minutes of the video, $[(\tau - b)^+, \tau]$, and continuously caches the most recent content as time goes along. This client can serve any client requesting the same video starting at a position within $[(\tau - b)^+, \tau]$.

As the amount of cached stream, b, increases, the scalability of DirectStream improves (see Proposition VI.1). However, the value of b is restricted by (i) the available cache space at clients, and (ii) the willingness of clients to server other clients for an extra time after finishing watching the video themselves. Since the clients may have to forward the stream to other clients for b minutes after they finish watching the video, clients have a strong incentive to use a small value of b even if they have large storage space. In case clients leave immediately after playing back, the clients may have to switch to new parents at the end of video playback. In that case, more dedicated resources, such as servers, have to be deployed.

B. Using AMDirectory service

AMDirectory service maintains information regarding content servers and clients to facilitate the search by a new client for a parent client. A user community is created for each video in AMDirectory service. Content servers immediately register themselves with AMDirectory and indicate themselves as servers. AMDirectory service maintains one entry for each client. For instance, the entry for client i is a tuple (a_i, t_i, τ_i, b_i), where a_i is client i's IP address, t_i is the time when this client started to receive the stream, τ_i is the position in the video where the client began, and b_i is the client's buffer size. At any future time t, the content cached at client i is $\Big[\min\{L-b_i, \tau_i+(t-t_i-b_i)^+\}, \min\{L, \tau_i+(t-t_i)\}\Big]$, where L is the video length. The implementation of AMDirectory service is described in Section IV.

C. Parent client selection

The peer clients in DirectStream form peer-to-peer overlays over which the video stream is forwarded. Since a minimum guaranteed bandwidth equal to the playback rate is required over all connections in the streaming overlay, a client has to select a parent node that has sufficient bandwidth to itself. Moreover, the selection of the parent node should allow the DirectStream to admit as many clients as possible in the long run.

We propose QoS peer selection that mimic the QoS routing at the application level. Assume that client A sends a service request to AMDirectory service and receives a list of n candidates, $\{c_i\}_{i=1}^{i=n}$, that can provide the service. Suppose that the measured number of hops from candidate client c_i to the requesting node is n_i, and the

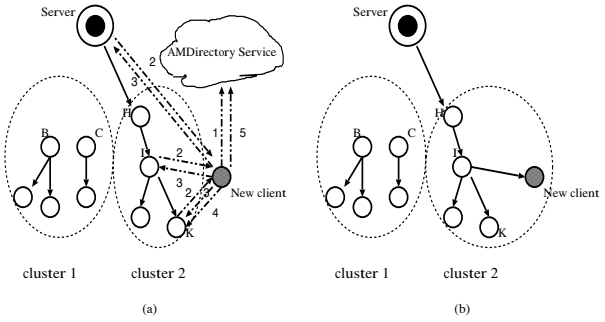


Fig. 2. Servicing new clients in DirectStream system. (**a**) A new client arrives at system. The dashed lines represent the signaling paths. (**b**) The new client selects client I as its parent and starts to receive the stream from client I.

measured available bandwidth is x_i. The QoS peer selection algorithm uses the *distance-bandwidth ratio*, defined as n_i^r/x_i where r is a positive constant, as the metric in selecting the peer peer client. The new client selects the candidate with smallest distance-bandwidth ratio to be its parent client. Intuitively, the selection of a parent with large available bandwidth helps to balance the workload; while the selection of a parent close by reduces the traffic placed on the underlying network. QoS peer selection tries to properly balance the above two factors.

Routing traffic with bandwidth guarantees has been studied extensively in the QoS routing context. However, the current Internet has not fully supported the QoS routing yet. QoS peer selection attempts to mimic QoS routing at the application level so as to maximize the number of clients that can be serviced in the long run. Our preliminary results in [35] show that the QoS selection algorithm performs better than traditional selection algorithms, such as shortest-widest selection algorithm and widest-shortest selection algorithm [36].

D. Servicing a new client

Assume that a new client wants to watch video A starting at position τ. The service search process for such a request consists of the following five steps, as indicated in Fig. 2(a).

Step 1. The new client initiates the AMDirectory lookup service, asking for video A starting at position τ.

Step 2. The AMDirectory lookup service allows the content servers and the clients that can serve this request to send response messages to the requesting client. The lookup process in AMDirectory service is presented in Section IV-A.

Step 3. The new client selects the parent client using the QoS parent selection algorithm.

Step 4. The new client contacts the selected parent client and asks it to forward the stream.

Step 5. After successfully setting up the connection, the new client registers itself in the AMDirectory.

In the example illustrated in Fig. 2, the new client selects client I as its parent and joins the cluster. It will receive the stream from client I thereafter.

A similar procedure can be used to support *client recovery* and *VCR functionalities*. In peer-to-peer video streaming, a client may leave the system without advanced notice. Its children clients lose their parent and cannot receive the stream any more. In this case, the children clients start the recovery process. The recovery process is the same as a new client joining process except that the video starting position will be at the parent client's departure point.

IV. AMDirectory Service: Application-level Multicast based Directory Service

A directory service allows users to locate other users sharing the same interests. It is an essential building block for many distributed applications. For instance, in peer-to-peer video streaming, an incoming client uses the directory service to identify other clients that are watching the same video and are willing to support incoming clients as peers.

We propose a new directory service, AMDirectory service, to dynamically distribute and track the location of users in the network. Traditional directory services are implemented using a central server, which is a single point of failure and can become the performance bottleneck. AMDirectory service employs application-level multicast as the basis for scalable directory service. For each community sharing the same interest, an application-level multicast group is created. Users register themselves by joining the multicast group, and unregister themselves when leaving. The lookup service is provided to allow users to check out other users in a group. Compared to a centralized directory server, AMDirectory service has the following advantages:

- *Avoiding the single point of failure problem.* The mechanism built into application-level multicast allows AMDirectory service to recover even if some nodes go down.
- *Scalable*. An application-level multicast group can handle more information than a single directory server.
- *Providing proximity information*. Proximity information is vital in constructing an appropriate peer-to-peer overlay for a streaming service. Since some application-level multicast trees are built taking proximity into account, AMDirectory service can use them to provide the approximate proximity information naturally.

AMDirectory can be implemented using any application-level multicast. We have chosen to implement it on top of Scribe [20] because Scribe builds a proximity-aware spanning tree on top of Pastry [19], a structured peer-to-peer network. It supports both small

and very large groups, and it allows nodes to join and leave a group with low overhead.

A. Design of AMDirectory service

AMDirectory service is built on the top of Scribe, and supports the following operations: community creation, user registration, user un-registration, and user lookup service. In the following, we describe these operations, respectively.

• *User community creation.* AMDirectory service uses one multicast group for each user community. The *communityId* of this user community can be the hash of this community's unique name, e.g., the video title concatenated with its creator's name. A user community is created by creating a Scribe multicast tree [20].

• *User registration and un-registration.* User registration and un-registration in AMDirectory use the underlying Scribe node join and departure operations. However, there may be multiple users attached to the same node. When a user registers with AMDirectory service, the corresponding Pastry node records the user's information, and checks whether it is already in the corresponding Scribe multicast tree. If it is, nothing needs to be done. Otherwise, it initiates the Scribe join process.

When a user unregisters from AMDirectory service, the underlying node take this user's information off the record. If this user is the last registered user at this node, the node leaves the Scribe multicast tree.

• *User lookup service.* User lookup is the key service provided by AMDirectory. It allows a user to identify other users in a community that satisfy certain criteria, denoted as *lookup criteria*. For instance, a user may determine all users that are willing to serve the video. The lookup service can be implemented by extending the available Scribe APIs and such an extension is straightforward. Below we describe the procedure of user lookup without getting into its implementation details.

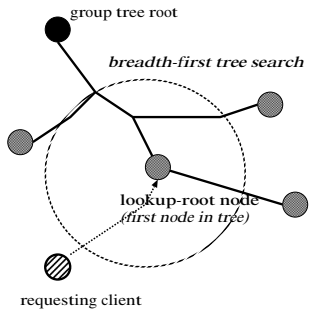


Fig. 3. User lookup in AMDirectory service. An incoming user send a *lookup* message toward the tree root. The first node in the tree intercepts the message, and conducts a breadth-first search.

A client that intends to perform the lookup service first obtains the community id of interest. The community id can be retrieved from a public web site, or through other bootstrapping methods (for instance, the community id can be the hash value of a video title). The client then sends a lookup message toward the root of this community's group tree, as shown in Fig. 3. We call the first node that is in the Scribe tree and on the route from the requesting client to the tree root the *lookup-root node*. A breadth-first search is then conducted on the tree with the lookup-root node as the root (rather than the original tree root). More specifically, when a node in the group tree receives a lookup message and determines it is the lookup-root node (this can be realized by setting a flag inside the lookup message), it initiates the breadth-first search. The lookup-root node first informs the local registered users of the lookup criteria. It then forwards the lookup message to its children and parent nodes in the Scribe tree. When non-lookup-root nodes receive a lookup message, they also inform the local registered users of the lookup criteria. They then forward the lookup message to their children and parent nodes except the one from which the message is received.

The breadth-first search on the tree rooted at the lookup-root helps the requesting client estimate the proximity of responding users. The lookup message contains a depth field that is set to be zero at the lookup-root node. The depth increases by one each time the lookup message is forwarded to the next node. Upon receiving a lookup message, a registered user checks whether it satisfies the lookup criteria. If it does, a response message is sent back to the requesting client with the depth information. Since the lookup-root node is the first node that the lookup message comes across, it is treated as the closest node to the requesting node. In addition, the nodes having small depth are likely to be close to the lookup-root node, and hence close to the requesting client as well.

The proximity information of users is vital for many applications to improve the overall performance. In DirectStream, when the client request rate is high, there are thousands of concurrent users in the system. A new client can have hundreds of candidate parent clients. Selecting an appropriate parent from a large number of candidates can incur high overhead, and significantly delay the client's joining process. As shown in Section VI-B.7, we use AMDirectory service to select a small set of candidates, while still achieving performance similar to as if all candidates are considered.

To avoid the large number of responses flooding the requesting node, we can limit the searching scope. For instance, we can use a *TTL* (time to live) field in the lookup message so that the lookup message will not traverse beyond certain number of hops starting from the lookup-root.

V. Discussions – Supporting VCR operation and providing continuous playback

A. Supporting VCR operations

The support of VCR operations is straightforward in DirectStream. Below we describe how DirectStream can support jump forward, jump backward, and pause.

The clients invoking *jump forward* or *jump backward* want to play normally from an arbitrary point in the video. For instance, client A is currently playing the video at position τ_A with a buffer size of b_A, and would like to jump to $\tau_A^{'}$. If $\tau_A^{'} > \tau_A$, this is a jump forward operation. Otherwise, it is a jump backward operation.

1) Jump forward: The jump forward operation is handled differently based on whether the new start position, $\tau_A^{'}$, falls within the caching window of its parent client. If $\tau_A^{'}$ falls within its parent's caching window, then A can send a *shiftTo* $\tau_A^{'}$ command to its parent, asking it to forward the stream from the new position $\tau_A^{'}$. Client A will inform AMDirectory service of its new caching window. Since client A misses the stream from τ_A to $\tau_A^{'}$, all of its children clients need to find alternative parent clients. Client A sends *recover* messages to all of its children clients, who then start the recovery process.

On the other hand, if $\tau_A^{'}$ does not fall into the caching interval of its parent client, client A needs to use AMDirectory service with the new starting position $\tau_A^{'}$, and follows the same procedure as serving a new client as described in Section III. The child clients of A have to locate the alternate parent clients as well.

2) Jump backward: For the jump backward operation, there are also two scenarios: (1) the new position $\tau_A^{'}$ lies within the local caching interval; or (2) it does not. In the first case, A simply shifts the playback point. In the second case, the procedure is the same as the jump forward operation with the new position outside the caching interval of the parent client. The child clients of A have to locate the alternate parent clients as well.

3) Pause: Pause removes the client temporally from the system. The client unregisters itself from AMDirectory service, and sends a *recover* message to all of its children clients who continue the playback. When it resumes play back, it rejoins the DirectStream like a new client.

4) Other operations: Other operations, such as fast forward and fast backward, cannot be supported naturally in DirectStream. One way to address this issue is to treat fast forward and fast backward streams as separate streams. In such case, the users conducting fast-forwarding or fast-backwarding can share the stream among them using DirectStream.

B. Providing continuous playback

As in any P2P application, a participant can leave at any time without advanced notice. We denote this as the *client early departure*. Meanwhile, the available bandwidth varies over the time. Although the study in [37] indicates that the available bandwidth remains stable for a relatively long period of time, the playback will be interrupted if the available bandwidth becomes less than the required bandwidth. We denote this as *bandwidth starvation*. Both client early departures and bandwidth starvation disconnect the peer-to-peer overlay, hence the reconstruction of overlay is required. In DirectStream, we let the direct children of departing client contact AMDirectory service to find the new parent client. During this process, the downstream clients' continuous playback can be disrupted.

Data buffering has been widely used in combating bandwidth fluctuation. Buffering a short period of video content and delaying the start of playback is an effective means to deal with the short-term bandwidth starvation. It also allows a client to locate an alternative parent in case its current parent leaves the system. Furthermore, if a client's ancestor clients depart early, the above technique is also effective since the buffered data at the intermediary clients along the path from the ancestor to the current client can help to shield the disruption.

However, data buffering alone may not be enough to handle the peer dynamics. In the following, we introduce Multiple Forwarding Parents scheme that enables the tradeoff between system's scalability and clients' reception quality.

1) Multiple backup parents (MBP) scheme: trading off scalability against quality: Multiple backup parents scheme allows a client connecting to multiple parents. As shown in Fig. 4, client A has four parents. Among them, parent 2 is the active parent that serves the data. Parent 1, 3,4 are backups. A backup can become an active parent if parent 2 leaves the system and the backup parent has the bandwidth to accommodate one more stream. By contacting the backup peers, the client shortens the joining delay and gets reconnected quickly.

Such simple multiple backup scheme is not new however effective. Skype [38] employs the similar scheme with multiple relay points. The measurement study shows that Skype offers reasonably good quality to users. Unlike voice service, though, the video streaming demands higher bandwidth. A backup without redundant bandwidth does not help.

To address this issue, MBP lets individual client reserve the amount of bandwidth of one stream for backup purpose. For instance, in the right side of Fig. 4, client B reserves one stream bandwidth, and is the backup parent of three clients. In case any of these clients need to locate a new parent, the first one who made the request will be able to use client B as its new parent.

Assume that a client has m backup parents, and each backup parent accommodates up to n clients as their backup parent. Suppose that a client loses its parent with probability p, then the probability of using one of

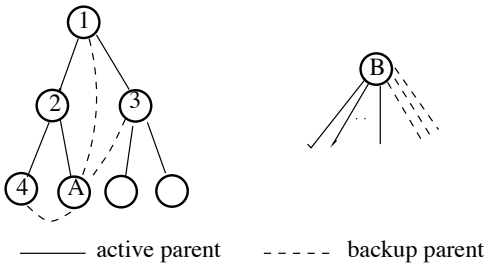


Fig. 4. Multiple backup parents (MBF) scheme. Left side is a simple example with client A with one active parent and three backup parents. Right side shows client B serves as active parents as well as backup parents. It reserves the bandwidth of one stream for all backup clients.

its backup parent as new parent can be approximated by $P = (1 - p/m)^{n-1}$; and the probability that this client successfully finds new parent from the backup parent list is $1 - (1 - P)^m$.

If a client has less than m backup parents, it periodically contacts the directory service to retrieve more candidates. The use of MBP together with data buffering significantly improves clients' continuous playback and reduce the chance to be ejected out of the system due to the peer churn, as shown by the simulation in Section VI-B.8.

VI. Performance Evaluation

In this section, we examine the performance of DirectStream via analysis and simulation. We derive closed form formulas for the average workload placed at the content server, and assess the impact of VCR jump-forward/jump-backward operations on the content servers. We conduct extensive simulation experiments to evaluate the performance of DirectStream, paying particular attention to the occasions where clients behave non-cooperatively. The results show that DirectStream can significantly reduce the workload imposed on the server, and scales extremely well as the popularity of the video increases even if the participating clients behave non-cooperatively. Moreover, simulation shows that AMDirectory service allows DirectStream to perform well even in the face of a large number of concurrent users. The incoming new client can quickly identify the suitable parent client without searching through the entire candidate parent list that may contain hundreds of candidates.

A. Performance analysis

1) Average server stress: We define the server stress at time t, $S(t)$, to be the number of streams sent out from the content server. In DirectStream, the content server only needs to stream a single copy to the first client in a cluster. All other clients within the cluster obtain the content from their respective parent clients (see Fig. 1). In the following analysis, we assume that the request arrival process is Poisson with an arrival rate of λ, and that all clients have the same size buffer of b minutes. In addition, we assume that a client has sufficient upload bandwidth to support as many peer clients as required. If a client can receive the content from a peer client, it does so rather than obtain the video from the server.

We denote by $E[S]$ the average server stress. Let L be the video length, W the normalized workload ($W = \lambda L$), and ρ the effective buffer size ($\rho = b/L$). We have the following Proposition.

Theorem VI.1: Consider DirectStream with a single content server. If the arrival process is Poisson with the rate λ, then

$$E[S] = We^{-\rho W}. \quad (1)$$

Furthermore, on average, each client serves $1 - e^{-\rho W}$ clients.

The proof is included in the Appendix.

Remark: First, the average server stress reaches the maximum of $(e\rho)^{-1}$ at $W = 1/\rho$. For instance, if the cache size is larger than 5% of the video length, i.e., $\rho \geq 0.05$, the average server stress does not exceed 8 streams. Secondly, Figure 5 plots the average server stress for different values of ρ, and compares them with the lower bound on server stress for IP-multicast based schemes, which is $\ln(1 + W)$ as derived in [39]. We observe that the average server stress in the DirectStream decreases as the normalized workload increases once $W \geq 1/\rho$, while the lower bound for an IP-multicast based scheme continues to increase logarithmically.

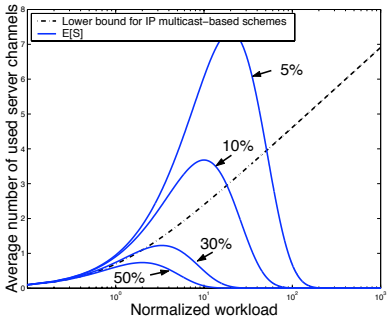


Fig. 5. Average server stress vs. normalized workload with the buffer size of 5%, 10%, 30%, and 50% of video length

2) Average server stress with freeloaders: Now we study the effect of clients that do not behave cooperatively. We call clients that receive the DirectStream service however refuse to serve other clients *freeloaders*. Assume that a client is a freeloader with probability p, we have the following results.

Theorem VI.2: Consider a DirectStream system with a single server. The arrival process is Poisson with the rate λ. If a client is a freeloader with probability p, then:

$$E[S] = We^{-(1-p)\rho W}. \quad (2)$$

Furthermore, on average, a non-freeloader client supports $\frac{1-e^{-(1-p)\rho W}}{1-p}$ clients.
The proof is included in the Appendix.
Remark: First, note that freeloaders reduce the effectiveness of data buffering. For instance, the average server stress with no freeloaders and 5% video length cache is equal to the average server stress where 50% of the clients are freeloaders and the buffer can store 10% of the video. Second, the maximum server stress is $1/(e\rho(1-p))$ when $W = 1/(1-p)\rho$. Finally, each non-freeloader has to serve more clients with freeloaders than without freeloaders since $(1-e^{-(1-p)\rho W})/(1-p) \geq 1-e^{-\rho W}$. This can cause a concentration of network traffic at clients and thus reduce the system's scalability, as shown in the experiment in Section VI-B.4.

3) Effectiveness in supporting VCR operations: We further look into the effectiveness of DirectStream in supporting VCR operations, described in Section V-A. We hope that VCR operations can be supported with little impact on the servers. More specifically, we are interested in the probability that a VCR request has to be serviced by the content server rather than by a peer client. If this probability is small, we infer that DirectStream can support VCR operations with small impact on the servers.

Assume that there are no freeloaders and all clients watch the video from the beginning. For such a DirectStream system, suppose there is an *extra client* (who doesn't belong to the arrival process) that arrives at an arbitrary time t and wants to watch the video starting at an arbitrary point τ within the video. Let P be the probability that this extra client has to obtain the service from the server. We have the following result.

Theorem VI.3: Consider a DirectStream system with a single server. The arrival process is Poisson with the rate λ, and clients watch the video from the beginning. Then:

$$P \leq e^{-\rho W}. \tag{3}$$

The proof is included in the Appendix.
Remark: Notice that P decreases quickly as the buffer size and the normalized workload increases. Intuitively, each client in DirectStream caches b minutes of video, and the aggregate of these intervals covers most of the video when the arrival rate is high. So a VCR jump forward or backward operation can be served with high probability by a client whose cached interval cover the required starting point.

B. Simulations

1) Simulation settings: We evaluate DirectStream using a network topology generated by GT-ITM [40] with 100 nodes. It consists of one transit network (with 4 nodes) and 12 stub domains. We assume that each node represents 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. We further assume that one node at each local network joins Pastry overlay, from which AMDirectory service can be accessed. Clients in the same local network use this node as a proxy to receive AMDirectory service.

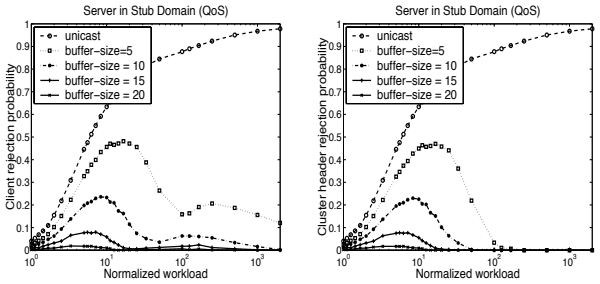
We assume a constant bit rate (CBR) video playback. We assign a bandwidth to each link in terms of the number of playbacks a link can support. The capacities of links between transit nodes and between transit nodes and stub domain nodes are chosen to be larger than those between stub domain nodes since links in the core network are typically better provisioned and have more bandwidth than the edge links. Using advanced coding techniques, videos with the playback rate from 300bps to 500bps offer reasonably good viewing quality. A link with the capacity of 100Mbps can support 200 to 333 such streams. Since we simulate DirectStream providing service for one video, we choose the capacity of each core link to be 10, i.e., core links can support up to 10 streams simultaneously; and that of each edge link to be 3 for the simulation results reported in this paper. The video length is set to 100 mins, and the parameter r in the QoS selection algorithm is set to 0.5.

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 in the network. 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.

2) Client rejection probability: We first place the server at one of the stub nodes. Fig. 6(a) depicts the rejection probability vs. the normalized workload, W, for unicast (client-server service model) and DirectStream with different buffer sizes, ranging from 5% to 20% of the video length. The normalized workload is the product of client arrival rate and video length. Compared to the traditional client-server service model, DirectStream significantly serves more clients, especially when the client request rate is high. Furthermore, we observe that the rejection probability for DirectStream exhibits an interesting bimodal behavior, with one mode in a low workload region and the other in a high workload region.

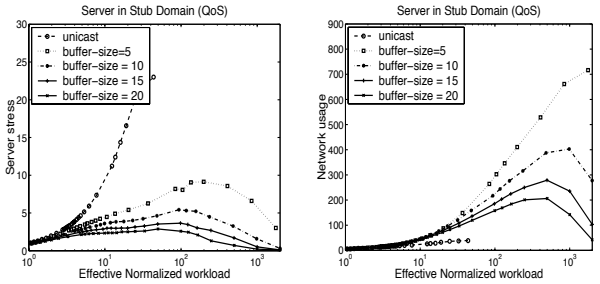
We argue that the first mode is due to limited bandwidth at the server, while the second one is caused by a combination of limited bandwidth at the server and limited capacity of the entire network. We denote by *cluster headers* the clients whose candidate list only contains the content server. Fig. 6(b) shows the rejection probability caused by the cluster headers that intend to obtain the video from the server directly. Comparing the two graphs in Fig. 6, we observe that the rejection of the cluster headers by the server mainly contributes to the first mode. Proposition VI.1 also indicates that when the load equals $1/\rho$, the server bandwidth requirement reaches the maximum, which coincides with the maximum rejection probability in Fig. 6(b).

Regarding the second mode, as the client request rate



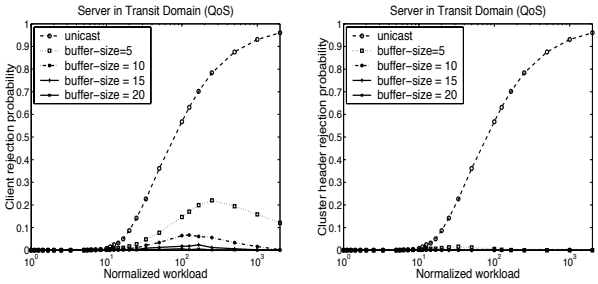
(a) Rejection probability (b) Cluster headers' rejection probability

Fig. 6. Overall rejection probability and cluster header rejection probability in DirectStream with the server in the stub domain



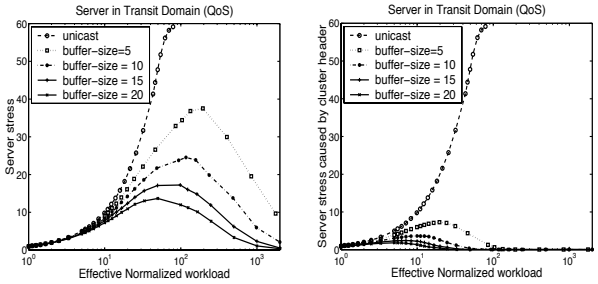
(a) Server stress (b) Workload on the entire network (network usage)

Fig. 7. Server stress and network usage in DirectStream



(a) Rejection probability (b) Cluster headers' rejection probability

Fig. 8. Overall rejection probability and cluster header caused rejection probability in the DirectStream with server in transit domain



(a) Server stress (b) Server stress caused by cluster headers

Fig. 9. Overall server stress and cluster header caused server stress in DirectStream with server in transit domain

increases, the server stress (Fig. 7(a)) and network usage (Fig. 7(b)) also increase. Thus client requests are more likely to be rejected due to limited available bandwidth at the server as well as between clients. However, as the workload further increases, the likelihood of a nearby peer increases. Hence the rejection probability decreases as the arrival rate further increases.

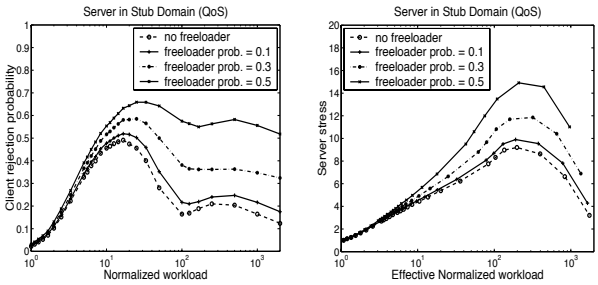
3) Effect of server bandwidth: We investigate the effect of the server bandwidth by placing the server at one of the transit nodes, where more bandwidth is available.

First we notice that the rejection probability in Fig. 8(a) is unimodal rather than bimodal as illustrated in Fig. 6(a). Fig. 8(b) depicts the rejection probability of cluster headers. The rejection probability caused by cluster headers is almost zero over the entire workload range. This can be explained by the abundant bandwidth present at the server because of its location in stub domain. The maximum average server bandwidth required is $(e\rho)^{-1}$, and is less than the link bandwidth of 10 in transit domain. Thus the server is not a bottleneck and the rejection probability due to the limited server bandwidth is eliminated.

In terms of server stress caused by the cluster headers (see Fig. 9(b)), again it is consistent with our previous analysis, namely reaching the maximum at $1/p$. Furthermore server stress decreases as the workload increases further.

4) Effect of freeloaders: Here we investigate the effect of freeloaders on the performance of DirectStream. As depicted in Fig. 10, the rejection probability and the server stress increase as the freeloader probability increases. However, even when the freeloader probability is high, the rejection probability exhibits a similar trend as it does with lower freeloader probability, consistent with Proposition VI.2.

In further examining the effect of freeloaders, we compare the following two scenarios: (1) buffer size equal to 5% of the video with no freeloaders; and (2) buffer size equal to 10% of the video length with the fraction of free loaders equal to 0.5. According to Proposition VI.1 and VI.2, the server stress is the same for both scenarios. Fig.



(a) Client rejection probability (b) Server stress

Fig. 10. Performance comparison with freeloaders

11 compares their rejection probabilities. When the normalized workload is low, the rejection probabilities are close. However, as the normalized workload increases, the system with larger client buffers and larger fraction of freeloaders exhibits significantly more rejections. According to Proposition VI.2, each non-freeloader has to support $(1 - e^{-(1-p)\rho W})/(1 - p)$ children clients. So the non-freeloaders in the second scenario support more clients than in the first scenario. This makes it difficult to balance the workload, and leads to a high rejection probability when the normalized workload is high.

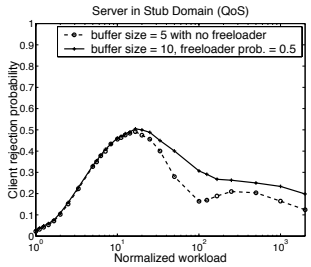
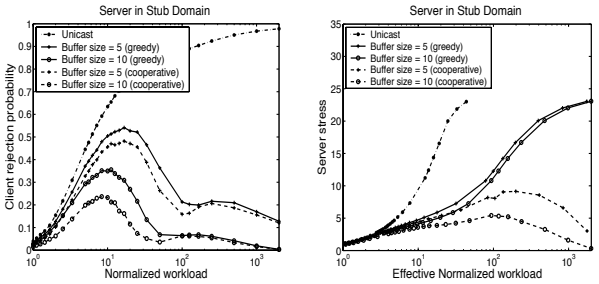


Fig. 11. Performance comparison of the case with buffer size equal to 5% with no freeloaders and buffer size equal to 10% with freeloader probability 0.5.

5) Effect of greedy clients: Another type of non-cooperative client is the *greedy client*. A greedy client connects to the server directly whenever possible since the server will not depart in the middle of streaming and thus does not impose the *client early departure problem*, as defined in Section V. Fig. 12(a) compares the rejection probability for a system with cooperative clients to that for a system with greedy clients. Although the rejection probability for a system with greedy clients is higher than that with cooperative clients, it still exhibits a similar shape, and scales well as the request rate increases. Intuitively, the server can be modeled as a $M/D/C/C$ queue, where the service time is the video length, and C is the number of channels available at the server. In a cooperative environment, each server channel is used to serve a cluster; while in the non-cooperative environment, the greedy clients try to grab an idle channel whenever possible. However, the greedy clients are forced to form a cluster when all channels are occupied. This explains why the rejection probability curves exhibit the similar shapes. As to the server stress (see Fig. 12(b)), DirectStream with greedy clients imposes a much greater workload on the server, which does not decrease even when the request rate is high due to the greediness of clients.

6) Random buffer size: In the experiments presented above, we assume that the clients' buffer are of the same constant size. Fig. 13 depicts the client rejection probability with random buffer size, where the buffer size is uniformly distributed around its average size, with the deviation equal to half of the average size. The rejection probability with constant and random buffer size are close to



(a) Client rejection probability (b) Server stress

Fig. 12. Performance comparison of cooperative and greedy clients.

each other, suggesting that the randomness of buffer size does not have a significant impact on the system's performance.

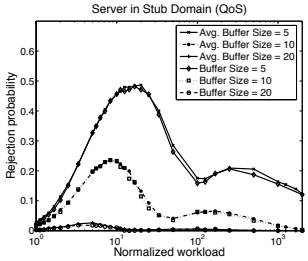
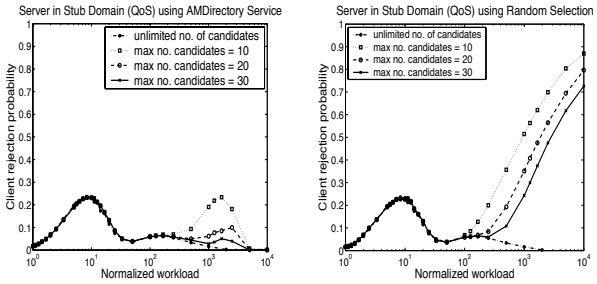


Fig. 13. Rejection probability with constant and random buffer size

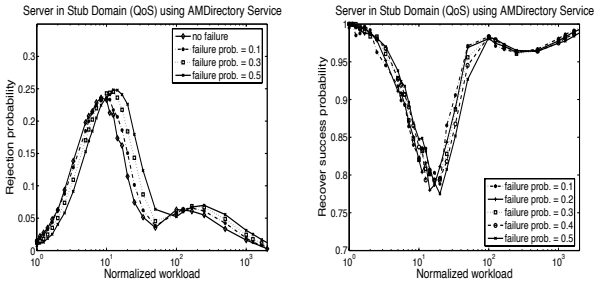
7) Efficient parent client selection using AMDirectory service: As the client request rate increases, the number of concurrent clients in the system increases. A new client has many candidate parent clients from which to select the right parent. Since a certain amount of measurement work has to be done for each candidate, a large number of candidates could significantly slow a new client's joining process, and impose substantial measurement overhead on the network. In the following, we show that AMDirectory service allows a new client to sample a small number of candidates, and that DirectStream still achieves good scalability.

Fig. 14(a) depicts the client rejection probability when a new client is only allowed to contact a limited number of candidates. A new client limits the number of candidates by limiting the searching scope combined with actively dropping the candidates that are further away from it (by comparing the depth value, see Section IV). The maximum number of allowed candidates are 10, 20, and 30, respectively, for the results presented in Fig. 14(a). Note that the rejection probability is very close to the rejection probability using all possible candidates when the normalized workload is low (< 300). When the normalized workload is between 300 to 3000, the rejection probability increases when the number of candidates decreases. This is due to the fact that although AMDirec-



(a) Using AMDirectory service (b) Random selection

Fig. 14. Performance comparison with/out AMDirectory service using limited no. of candidates.



(a) Rejection probability (b) Recover success probability

Fig. 16. Performance comparison with/out client failure

tory service returns a list of close-by clients, it does not consider the available bandwidth information. Therefore, some candidates with large available bandwidth but further away form the requesting client can be ignored by AMDirectory. However, as the workload increases further, the proximity information plays a more significant role in selecting parent client. The rejection probability with limited candidates thus gradually decreases.

We further compare the client rejection probability using AMDirectory service to that using random selection algorithm, where a small set of candidates are selected randomly among all possible candidates. Fig. 14(b) depicts the results using random selection. We observe that the rejection probability using random selection is larger than that using AMDirectory service, and continues to increase and eventually blows up. The rejection probability using AMDirectory service decreases beyond certain point, hence the scalability is still achievable in DirectStream with limited number of candidates using AMDirectory service.

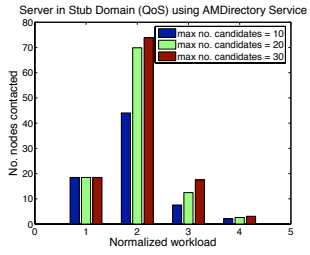


Fig. 15. Number of nodes contacted to find enough candidates (X-axis is the exponent of normalized workload with base of ten)

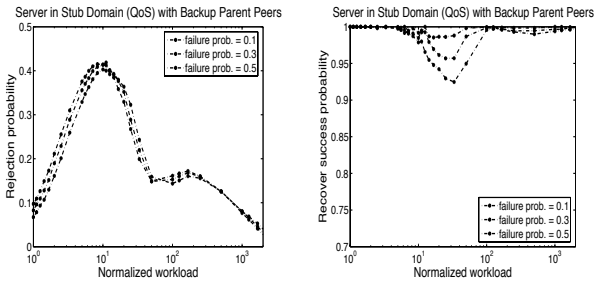
Finally, we examine how fast the AMDirectory's lookup service is. We use the number of nodes that are contacted during the search process as the metric. Note that the same metric also reflects the amount of messages passed in the network during AMDirectory lookup service. Fig. 15 depicts the number of nodes contacted vs. the normalized workload with different maximum number of allowed candidates. The X-axis is the exponent of the normalized workload with base of ten. The buffer size is chosen to be 20% of the video length. The conclusions drawn from this example hold for different buffer sizes.

When the normalized workload is low, the total number of possible candidates is less than the maximum number of allowed candidates. Hence, the entire multicast tree is traversed to find all possible candidates. This is obvious with normalized workload of 10 where the same number of nodes are contacted regardless of the maximum number of allowed candidates. In contrast, when the normalized workload increases beyond the normalized workload of 100, the number of contacted nodes decreases significantly. For instance, when the normalized workload is 1000, the number of contacted nodes is in the range of 7 to 18 nodes.

We also like to point out that the majority of these contacted nodes are encountered during the breadth-first tree search phase (see Section IV-A), which means that these nodes are come across in parallel because the lookup service proceeds along the tree branches parallelly. This further reduces the lookup time.

8) Impact of client failure: As described in Section V-B, a participant can leave at any time without advanced notice due to failure. In DirectStream, the immediate child(ren) of the departing client will initiate the recover process once it detects the parent's departure. The further downstream descendent clients wait for the outcome. A client starts the recover process only if its immediate parent client leaves the system.

We investigate the system performance in the face of client failure. We assume that a client fails in the middle of the playback with certain probability, and the departure point is uniformly distributed within the video length. A client that fails to recover is counted as a rejection in this experiment. Fig. 16(a) presents the rejection probability with and without client failure, where the buffer size is 10% of the video length. Unexpectedly, the failure only marginally affects the client's overall rejection probability. Intuitively, the rejection probability is the function of normalized workload and overall system capacity. Although the client failure introduces churn into the system and clients may experience un-continuous playback, it



(a) Rejection probability using MBF scheme (b) Recover success probability using MBF scheme

Fig. 17. Performance using MBF scheme

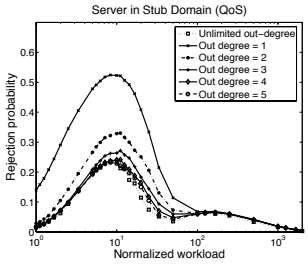


Fig. 18. Client rejection probability with various out degrees (out degree is the number of children a client can support)

does not change the overall system capacity significantly. This may explain the marginal change of client rejection probability.

Fig. 16(b) depicts the recover success probability vs. the normalized workload. We define the recover success probability to be the probability that a recovering client successfully find an alternative parent. The recover success probability is always greater than 0.75, and it converges to one as the normalized workload increases. Furthermore, the curve exhibits two dips, matching the two modes in the rejection probability. The experiment suggests that the DirectStream handles failure well, and the performance improves as the normalized workload increases.

We further look into the system performance with multiple backup parents (MBF) scheme in place. We set the number of backups to be 5; and each client serves 5 clients as their backup parent. Comparing Fig. 16(a) with Fig. 17(a), it is obvious the p2p system with MBF admits less clients. This is due to the bandwidth reservation for backup parents. Fortunately, the system still scales as the arrival rate increases further.

The recover success probability using MBF increases significantly comparing to the results without using MBF scheme as shown in Fig. 16(b). Even with 50% of users depart early, the clients still can recover with probability greater than 90%. We believe that 90% of success rate is pretty good comparing with the measurement study conducted in [41]. The study shows that PPLive, a popular p2p streaming service, is able to attract a large population of viewers even if between 10% to more than 50% of viewers experience playback freeze or reboot in a one hour time window.

9) Impact of client out degree: In the previous experiments, we have assumed that a client in the DirectStream puts no limit on the number of children client it is willing to forward the stream to. We define the number of children that a client is willing to support to be the *out-degree* of this client. In practice, a client may have limited out-degree. Here, we investigate how the out-degree impacts the client rejection probability.

Fig. 18 depicts the rejection probability against the normalized workload with different out-degrees. When the normalized workload is in the low to medium range, the more out-degree clients support, the lower rejection probability is. This is reasonable since more out-degree gives the DirectStream more flexibility to form better overlays. However, as the normalized workload increases further, the impact of the out-degree diminishes. This is due to the fact that clients only support a small number of children when the workload is high, as demonstrated by Theorem VI.1.

Another observation is that the out-degree plays an important role when its value is small, e.g. out-degree equal to one or two. Once the out-degree is greater than two, the rejection probability quickly improves and becomes close to that with unlimited out-degree. In practice, if peer clients can allow more than two children, we expect the out-degree will not impact the overall performance significantly.

VII. Conclusion and Future Work

In this paper, we propose DirectStream streaming media service that can efficiently and cost-effectively provide video on-demand service with VCR operation support. We show, through both analysis and simulation experiments, that DirectStream significantly reduces the workload posed on the server as well as on the network, and scales much better than the traditional client-server unicast service model even when the participating clients behave non-cooperatively. In addition, AMDirectory service allows DirectStream to perform well even in the face of a large number of concurrent users. The incoming new client can quickly select the appropriate parent client without searching through all possible candidates. Our results suggests that peer-to-peer networking is an effective technique to address the scalability issue in VoD service.

Future research can proceed along several avenues. As described in Section V, providing continuous playback in face of clients' early departure and bandwidth starvation is a challenging research issue. Secondly, the current design of DirectStream handles the non-cooperative clients passively. We are interested in developing a mechanism to encourage clients to cooperate with each other, and to

achieve certain fairness among peers. Finally, current DirectStream is designed with supporting CBR videos in mind. How to extend it to support VBR is an interesting research question.

APPENDIX

Proof of Theorem VI.1: Consider the client arrival process as depicted in Fig. 19. Client 0 is the first client and becomes the cluster header. For client i, $1 \leq i \leq n$, we assume that the inter-arrival time between i-th and $(i+1)$-th client, v_i^{i-1}, is less than b, i.e., $v_i^{i-1} \leq b$. Hence client i can obtain the stream from client $(i-1)$ and a cluster is formed among these clients. We further assume that $v_{n+1}^n > b$, then a new cluster is created by client $n+1$.

Define *the cluster inter-arrival time* to be the inter-arrival time between two adjacent cluster headers. Denote by R the cluster inter-arrival time. We have $R_0 = v_{n+1}^0$. The arrival of cluster headers forms a renewal process, and $\{R_i\}_{i=0}^{\infty}$ is an i.i.d. sequence. Since each cluster only retrieve one copy of video from server, we have:

$$E[S] = L/E[R]. \tag{4}$$

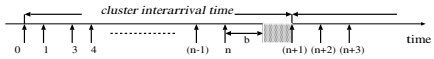


Fig. 19. DirectStream service: clients, clusters, and cluster length

Denote by q the probability that the client inter-arrival time is larger than b. Let $E[X]$ be the average client inter-arrival time given that the client inter-arrival time is smaller than b, and $E[Y]$ be the average client inter-arrival time given that the inter-arrival time is greater than b. We have:

$$E[R] = \sum_{m=0}^{\infty} (1-q)^m q \Big[mE[X] + E[Y] \Big]. \tag{5}$$

Since the client arrival process is Poisson with arrival rate λ, we have:

$$\begin{aligned} q &= e^{-\lambda b}. && (6)\\ E[X] &= \int_0^b x f(x|x<b)dx, \\ &= \int_0^b x \frac{\lambda e^{-\lambda x}}{1-e^{-\lambda b}} dx \\ &= \frac{1}{\lambda} - \frac{e^{-\lambda b}}{1-e^{\lambda b}} b. && (7)\\ E[Y] &= \int_b^{\infty} f(y|y>b)dy, \\ &= \int_b^{\infty} y \frac{\lambda e^{-\lambda y}}{e^{-\lambda b}} dy, \\ &= 1/\lambda + b. && (8) \end{aligned}$$

Substitute Equation (6)(7)(8) into Equation (5), we have:

$$\begin{aligned} E[R] &= \sum_{m=0}^{\infty} m(1-q)^m q E[X] + E[Y], \\ &= \frac{1-q}{q} E[X] + E[Y], \\ &= \frac{e^{\lambda b}}{\lambda}. && (9) \end{aligned}$$

Substitute Equation (9) into (4), we have:

$$E[S] = We^{-\rho W}. \tag{10}$$

where L is the video length, W is the normalized workload ($W = \lambda L$), and ρ is the effective buffer size ($\rho = b/L$).

As for the average number of peer clients a client has to serve, if the inter-arrival time is less than b, the client serves the following client; otherwise no client needs to be served. Therefore the average number of peer clients that a client has to serve is $1 - e^{-\lambda b} = 1 - e^{-\rho W}$. ∎

Proof of Theorem VI.2: Since a client is a freeloader with probability p, the client arrival process can be splitted into two Poisson processes: one for non-freeloaders with rate $(1-p)\lambda$, and the other for freeloaders with rate $p\lambda$. In the following we use subscript n and f to denote the variables related to normal clients and freeloaders, respectively.

Because freeloaders do not support other clients, the cluster inter-arrival time is determined by the non-freeloaders. Define the *shaded interval* to be the time interval from b minutes after the arrival of the last non-freeloader in a cluster to the arrival of next cluster's header (see Fig. 19). The free loaders arrived during the shaded interval have to obtain the service from the server directly. Let N_f be the number of freeloaders that arrives at the shaded interval. The average number of server streams to serve one cluster is thus $1 + E[N_f]$, and the average server stress is:

$$\begin{aligned} E[S] &= \frac{L}{E[R_n]}(1 + E[N_f]), \\ &= (1-p)We^{-(1-p)\rho W}(1 + p/(1-p)), \\ &= We^{-(1-p)\rho W}. && (11) \end{aligned}$$

Denote by N the average number of peer clients a non-freeloader client has to serve, and by t_n the non-freeloaders' inter-arrival time. We have

$$\begin{aligned} E[N] &= E[N|t_n < b] + E[N|t_n > b], \\ &= (1+p\lambda)\Big(\frac{1}{(1-p)\lambda} - \frac{e^{-(1-p)\lambda b}}{1-e^{-(1-p)\lambda b}} b \Big) \end{aligned}$$

$$= \cdot(1 - e^{-(1-p)\lambda b}) + p\lambda b e^{-(1-p)\lambda b}, \\ = \frac{1 - e^{-(1-p)\rho W}}{1-p}. \quad (12)$$

■

Proof of Theorem VI.3: Let us first consider the *shifted scenario*, where the extra client arrives at time $t - \tau$ and wants to watch the video from the beginning. There exists a client that can serve such request unless $t - \tau$ falling into the shaded interval. Since $t - \tau$ is randomly selected, the probability that $t - \tau$ falls into the shaded interval is $E[\text{shaded-interval-length}]/E[R] = e^{-\rho W}$.

The shifted scenario is equivalent to the original case (where the extra client arrives at t and wants to watch the video starting at τ) if the video is infinitely long. For finite length video, there may exist a client that can serve the extra client even if $t - \tau$ falling into the shaded interval. Therefore $P \leq e^{-\rho W}$. ■

REFERENCES

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

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

[3] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2cast: Peer-to-peer patching scheme for vod service," in *Proceedings of the 12th World Wide Web Conference (WWW-03)*, May 2003.

[4] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "A peer-to-peer on-demand streaming service and its performance evaluation," in *Proceedings of 2003 IEEE International Conference on Multimedia & Expo (ICME 2003)*, July 2003.

[5] 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.

[6] J. Li, P. A. Chou, and C. Zhang, "Mutualcast: An efficient mechanism for content distribution in a peer-to-peer (p2p) network," Tech. Rep. MSR-TR-2004-100, Microsoft Research, 2004.

[7] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "Coolstreaming/donet: A data-driven overlay network for live media streaming," in *Proc. IEEE INFOCOM*, March 2005.

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

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

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

[11] A. Mahanti, D. Eager, M. Vernon, and D. Sundaram-Stukel, "Scalable on-demand media streaming with packet loss recovery," in *IEEE/ACM Transactions on Networking*, vol. 11, April 2003.

[12] 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.

[13] A. Bar-Noy, G. Goshi, R. E. Ladner, and K. Tam, "Comparison of stream merging algorithms for media-on-demand," in *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, January 2002.

[14] E.G. Coffman Jr., P. Jelenkovic, and P. Momcilovic, "Provably efficient streaming merging," in *Proc. of Web Caching and Content Distribution*, June 2001.

[15] S. Acharya and B. Smith, "Middle man: A video caching proxy server," in *Proc. NOSSDAV 2000*, June 2000.

[16] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet," in *Proc. IEEE INFOCOM*, April 2000.

[17] Y. Chae, K. Guo, M. Buddhikot, S. Suri, and E. Zegura, "Silo, rainbow, and caching token: Schemes for scalable fault tolerant stream caching," in *IEEE Journal on Selected Areas in Communications on Internet Proxy Services*, September 2002.

[18] X. Zhang, M. Bradshaw, Y. Guo, B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Amps: A flexible, scalable proxy testbed for implementing streaming services," in *Proc. NOSSDAV 2004*, June 2004.

[19] 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.

[20] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "Scribe: A large-scale and decentralised application-level multicast infrastructure," in *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, 2002.

[21] M. Kamath, K. Ramamritham, and D. Towsley, "Continuous media sharing in multimedia database systems," in *Proc. of 4th International Conference on Database Systems for Advanced Applications (DASFAA'95)*, April 1995.

[22] A. Dan and D. Sitaram, "A generalized interval caching policy for mixed interactive and long video enviroments," in *SPIE Multimedia Computing and Networking Conference*, January 1996.

[23] S. Sheu, K. A. Hua, and W. Tavanapong, "Chaining: A generalized batching technique for video-on-demand systems," in *Proc. IEEE International Conference on Multimedia Computing and Systems*, June 1997.

[24] D. A. Tran, K. A. Hua, and T. T. Do, "Layered range multicast for video on demand," in *Proc. International Conference on Computer Communications and Networks (IC3N'02)*, October 2002.

[25] S. Jin and A. Bestavros, "Cache-and-relay streaming media delivery for asynchronous clients," in *International Workshop on Networked Group Communication*, October 2002.

[26] Y. Cui, B. Li, and K. Nahrstedt, "ostream: Asynchronous streaming multicast in application-layer overlay networks," in *IEEE Journal on Selected Areas in Communications*, vol. 22, January 2004.

[27] A. B. Abhishek Sharma and I. Matta, "dpam: A distributed prefetching protocol for scalable asynchronous multicast in p2p systems," in *Proc. IEEE INFOCOM*, March 2005.

[28] M. Zhou and J. Liu, "Tree-assisted gossiping for overlay video distribution," in *Kluwer Multimedia Tools and Applications*, 2006.

[29] C. Partridge, T. Menedez, and W. Milliken, "Host anycasting service," in *RFC 1596*, November 1993.

[30] D. Katabi and J. Wroclawski, "A framework for scalable global ip-anycast (gia)," in *Proc. ACM SIGCOMM*, 2000.

[31] S. Bhattachargee, M. Ammar, E. Zegurra, N. Shah, and Z. Fei, "Application layer anycasting," in *Proc. IEEE INFOCOM*, 1997.

[32] Z. Fei, S. Bhattachargee, M. Ammar, and E. Zegurra, "A novel server technique for improving the response time of a replicated service," in *Proc. IEEE INFOCOM*, 1998.

[33] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scalable application-level anycast for highly dynamic groups," in *submission*.

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

[35] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "A directory-based peer-to-peer on-demand streaming service," Tech. Rep. UM-CS-Tech-Report, Department of Computer Science, University of Massachusetts Amherst, 2002.

[36] Q. Ma and P. Steeniste, "On path selection for traffic with bandwidth guarantees," in *Proc. International Conference on Network Protocols*, October 1997.
[37] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the constancy of internet path properties," in *Proceedings of the ACM SIGCOMM Internet Measurement Workshop (IMW '2001)*, November 2001.
[38] S. Ren, L. Guo, and X. Zhang, "ASAP: an as-aware peer-relay protocol for high quality voip," in *Proc. IEEE ICDCS*, 2006.
[39] D. Eager, M. Vernon, and J. Zahorjan, "Minimizing bandwidth requirements for on-demand data delivery," in *IEEE Transactions on Knowledge and Data Engineering*, Sep/Oct 2001.
[40] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOM*, April 1996.
[41] X. Hei, Y. Liu, and K. Ross, "Inferring network-wide quality in p2p live streaming systems," tech. rep., Polytechnic University, 2007.