Topology-Aware Peer-to-Peer On-Demand Streaming

Rongmei Zhang Ali R. Butt Y. Charlie Hu

Purdue University, West Lafayette IN 47907, USA
{rongmei,butta,ychu}@purdue.edu

Abstract. In this paper, we consider large-scale high-bandwidth on-demand media streaming in a dynamic and heterogeneous environment. We present MetaStream, a scalable and distributed content discovery protocol that enables clients across the Internet to self-organize into a topology-aware overlay network, in which they can "cache and relay" a stream among nearby peers. We present the design and implementation of MetaStream and show experimental results obtained in a large-scale emulated environment. Our evaluation shows that MetaStream distributes relaying load among the clients in a topology-aware manner, imposing low link cost to the underlying network and low streaming load on the media server. MetaStream is also highly resilient to node or network failures and is capable of quickly recovering the streaming quality at clients even at high failure rates.

Keywords: Peer-to-peer, on-demand streaming, overlay networks, topology-awareness

1 Introduction

In this paper, we consider the problem of on-demand media streaming to a large set of clients spread across the Internet. Compared to real-time media streaming or simultaneous file downloading, on-demand streaming is distinct in two aspects: (1) asynchrony, as clients may request for the media at different times; and (2) non-sequentiality, as different clients may access the media starting at different parts of the media object.

There exist two approaches to on-demand streaming. The first approach adopts IP multicast to serve multiple requests by a single stream. Due to the synchronous nature of multicast, clients either wait for the next scheduled multicast session at the cost of some start-up delay [9, 12], or participate in more than one sessions simultaneously [15, 8]. The second approach exploits caching of media objects at the proxies [20, 3, 21], or at the clients [23, 7, 14, 5, 6]. The media can be retrieved from the caches instead of from the streaming server.

The client-side "cache and relay" scheme has several major advantages. First, it is an application-level solution without any assumptions about the underlying network (i.e., IP multicast), and it does not require the deployment of specialized proxies. Receivers caches a small portion of the media content after playback and they collectively form a cooperative overlay network by streaming from each other's cache. Second, the capacity of the streaming overlay scales with the population of clients. It has been shown that this "grassroot" approach can defeat IP-multicast-based solutions in terms of server-side bandwidth saving [5]. Since clients also contribute streaming bandwidth to the streaming overlay, it has the self-scaling property of peer-to-peer file sharing systems. Third, each client can select the streaming source based on various QoS requirements. When there exist more than one candidates in the overlay network, the client can choose the one with the best performance metric, e.g., the lowest delivery delay.

A key challenge for the client-side "cache and relay" approach is *how to locate available streaming sources in the overlay network, given that clients may be widely dispersed over*

In *Proceedings of 2005 IFIP Networking Conference, Waterloo, Ontario, Canada, May 2-6, 2005.*

the Internet and they can leave or fail at any time. This calls for a content discovery service for streaming information update and lookup. A straightforward solution is to deploy a centralized server that keeps track of all the streaming sessions. Obviously this solution suffers from the bottleneck and single point of failure at the centralized server. In [5], the content discovery service is implemented by a number of clients acting as discovery servers. Specifically, each streaming session is registered with the content discovery service by contacting a subset of discovery servers. Likewise, each request is also mapped to a subset of the discovery servers. A client can find eligible streaming sources by querying the content discovery service. However, the mapping between streaming sessions or requests to discovery servers is performed through inconsistent hashing, i.e., the mapping is dependent on the number of available servers.

In this paper, we propose MetaStream, a fully decentralized and adaptive protocol for content discovery in large-scale high-bandwidth streaming. Since the volume and the physical distribution of streaming requests are usually beyond estimation, it is difficult to predetermine the scale and the deployment of a content discovery service. In MetaStream, all clients participate in the content discovery service and take on the responsibility of finding and choosing streaming sources without any requirement for external configuration or management. MetaStream is also adaptive to client population and network condition dynamics, and streaming paths can be refined over time to maintain and improve the streaming quality at clients. When a failure happens at a client or in the network, MetaStream can relocate the streaming source and restore the streaming session quickly. Moreover, MetaStream is oblivious to the encoding of the media object. MetaStream can be used with any data encoding schemes such as layered encoding [21] or multiple descriptor encoding (MDC) [10] to address the resource heterogeneity of streaming clients.

In addition, MetaStream also aims to minimize the streaming cost at the backbone network. Due to the high streaming rates of media objects, the streaming overlay should be network bandwidth efficient, and should avoid long streaming paths between clients. For the content discovery service, this translates into *topology-awareness*: for each streaming request, if there are multiple candidates capable of providing the media object, the one that is closest in terms of network distance should be chosen. MetaStream has inherent topology-awareness by clustering clients based on the network distances between them, and a streaming request is always satisfied by a nearby source, if it exists. Topology-awareness also contributes to the scalability of MetaStream.

In summary, MetaStream provides an efficient distributed content discovery service that allows clients to locate streaming sources in a topology-aware manner. As a result, clients organize themselves into a topology-aware overlay network. We have developed a prototype implementation of MetaStream and have performed an extensive evaluation of MetaStream running over 1,000 clients in an emulated Internet topology consisting of 5,050 routers. The experimental results show that MetaStream can satisfy client streaming requests with only minimal server-side bandwidth consumption and at the same time incurs low network link cost.

The rest of the paper is organized as follows. Section 2 gives a brief background on the “cache and relay” approach to on-demand streaming. Section 3 presents MetaStream, a topology-aware content discovery protocol for media streaming. Section 4 discusses the methodology for evaluating MetaStream and Section 5 provides detailed evaluation results obtained from a large-sale emulated environment. Finally, Section 6 reviews related work and Section 7 draws concluding remarks.

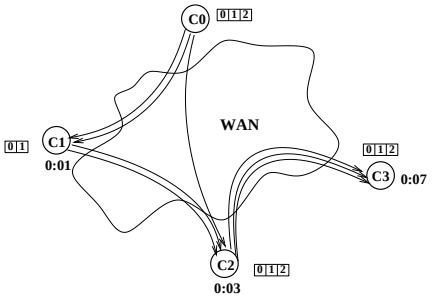

Fig. 1. Cache and relay

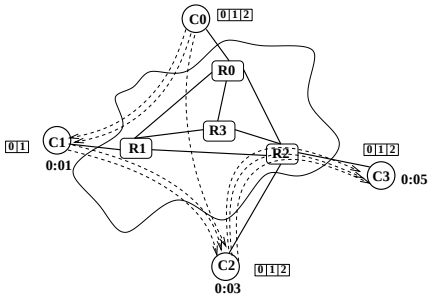

Fig. 2. Topology-aware cache and relay

2 Preliminaries

In the following, we give a brief overview of the client-side "cache and relay" scheme [5, 6, 14] for on-demand streaming.

For generality, we assume that the media object consists of one or more stripes, depending on if data encoding is applied. The object is originally provided by server C_0. The request from client $C_i (i = 0, \ldots, n)$ is characterized by the time T_i that C_i starts streaming and the offset S_i in the stream where the streaming begins. The achievable streaming quality at C_i is bounded by the input bandwidth. Each client also maintains a small cache of the media after playback and we denote the cache length as W_i. A later client C_j can stream from client C_i if their requests satisfy the following condition: $(T_i - S_i) < (T_j - S_j) < (T_i - S_i) + W_i$. The amount of stripes that a client C_i can relay to others is limited by its own output bandwidth. Any stripes that cannot be provided by the caches of other clients can be retrieved from the original server C_0.

A simple example of "cache and relay" is shown in Figure 1 where a stream is encoded into 3 stripes. Suppose that each client maintains a cache of 5 minutes and each starts the streaming from the beginning. The first client C_1 receives 2 stripes directly from the server C_0. The second client C_2 can retrieve the first 2 stripes from client C_1, but the third stripe has to be streamed from the server. When the third client C_3 arrives, its streaming request can be satisfied by client C_2.

Figure 2 gives an example of "topology-awareness" by showing a simplified network of routers connecting the streaming server and clients in Figure 1. If we assume client C_3 arrives 2 minutes after client C_2 and 4 minutes after client C_1, it is able to stream the first two stripes from either C_1 or C_2. However, it is more beneficial from the perspective of network bandwidth consumption to select C_2 since C_2 is only 2 hops away and C_1 is at least 3 hops away.

3 MetaStream

In this section, we present the design of MetaStream, a content discovery protocol for topology-aware on-demand streaming.

3.1 Topology-Based Clustering

In MetaStream, clients choose streaming sources based on the network distance. For this suppose, they self-organize into a dynamic hierarchy of clusters based on the network topology. In principle, any protocol for constructing a topology-aware hierarchy can be used. We use the NICE protocol [1] for the following reasons: (1) The source code is publicly available;

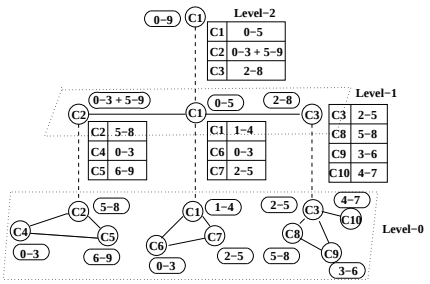

Fig. 3. Aggregation of cache state

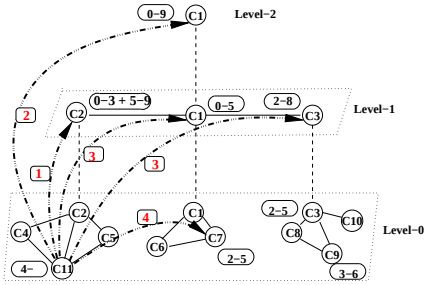

Fig. 4. Resolving streaming request

(2) The hierarchy can be built and maintained with very low overhead; (3) The hierarchy is adaptable to membership and network condition changes.

Due to page limitation, we omit the details of the NICE protocol. An example hierarchy of topology-based clusters as can be built using NICE is shown in Figure 3. Cluster leaders (i.e., cluster centers) of each level join the next higher level clusters, e.g., C_1, C_2 and C_3 of level-0 join level-1. By using a hierarchy of clusters, MetaStream can scale to a large number of clients.

3.2 Topology-Aware Content Discovery

To facilitate discovering nearby available streaming sources, a content discovery service is built on top of the topology-aware hierarchy.

Streaming State Registration and Aggregation First, streaming clients with available output bandwidth register with the content discovery service. Specifically, each client sends the information about the stripes being received and thus cached locally to its bottom level cluster leader. After aggregating the information from all the clients in the cluster, the bottom cluster leader sends a summary registration to the next higher level cluster leader. This registration process ends at the hierarchy root, which is the leader of the top level cluster.

The information presented to cluster leaders during registration is light-weight. Recall from Section 2 that client C_j can stream from client C_i if their requests satisfy $(T_i - S_i) < (T_j - S_j) < (T_i - S_i) + W_i$. Therefore, it suffices for client C_i to provide a range $(T_i - S_i, T_i - S_i + W_i)$ in the registration. Aggregation within each cluster is simply performed as a disjunction of all received registrations $\bigcup_i (T_i - S_i, T_i - S_i + W_i)$

An example of aggregation is shown in Figure 3. Each stripe of the stream is registered and aggregated separately. For clarity, only the aggregation for one stripe is shown in Figure 3. As a client may belong to multiple levels in the hierarchy, it may be responsible for receiving registrations and performing aggregations at each of these levels. If the hierarchy is created using the NICE protocol, each client maintains no more than $O(\log N)$ registrations from other clients.

Buffer registration is performed periodically and asynchronously by each client. Registration information is maintained as soft-state. The soft-state approach keeps registrations up-to-date as the hierarchy is undergoing all kinds of possible changes, such as when clients join and leave, clusters split or merge, and clients crash without notifying other members. In addition, MetaStream also invokes on-demand registration to react to changes in the composition of the stripes in a client's cache and its output bandwidth availability.

Resolving Request for a Single Stripe A client request for a streaming stripe is resolved by querying the content discovery service. Only the timing of the request, in the form of $(T_j - S_j)$, needs to be presented. At each step of resolving the request, potential streaming sources are identified by checking $(T_j - S_j)$ against the aggregated cache registration $\bigcup_i (T_i - S_i, T_i - S_i + W_i)$.

Figure 4 shows an example of resolving a streaming request based on the aggregation example in Figure 3. Suppose that a new client C_{11} wants to find a streaming source for stripe x. A request is first sent to the level-0 cluster leader C_2 (step 1). Generally, a request is forwarded up the hierarchy until it receives a successful response (step 2). Since higher levels have a wider view of the network, the request is more likely to be resolved. A successful response includes those next lower level cluster members that can further resolve the request. If the request is received at the root and the root is not aware of any other clients that can relay stripe x from their caches, server C_0 can accept the request and open a streaming session for the requested stripe.

After receiving one successful reply, the resolving process starts traversing the hierarchy downward (step 3), as it zooms in towards the closest streaming source. When the resolving process arrives at the bottom level and the closest candidate is determined, e.g., to be client C_7, client C_7 accepts the request if it confirms that it has enough output bandwidth remaining to relay stripe x to C_{11}; at this point the request is fully resolved.

Due to propagation delay along the hierarchy and the fact that aggregation partially relies on periodical refreshments, at some point in resolving the request, it may happen that client C_1 does not contain any matching record or client C_7 does not have enough remaining output bandwidth to accept C_{11}'s streaming request. When the closest candidate fails to further resolve the request, clients can continue to contact the next closest candidate. MetaStream also enables clients to back-trace to the previous levels of the hierarchy. This can be made possible by bookkeeping (e.g., using a stack) the results at each step of resolving the request.

Resolving Requests for Multiple Stripes In the descriptions of the content discovery service above, we consider only one stripe of the stream and make no assumption about the actual encoding scheme. Whether multiple description [**?**] or layered encoding [**?**] is used, the stripes can be resolved sequentially. As soon as one stripe is resolved, the stream is renderable at the client; the streaming quality improves as more stripes are available. In multiple description encoding, clients can choose from different combinations given the total number of stripes. In fact, clients just need to specify the desired number of stripes without defining which stripes. In contrast, in layered encoding, the order of the stripes to be resolved is fixed. Therefore, under comparable circumstances, multiple description coding allows more flexibility in satisfying clients' desired streaming quality.

It is worth noting that the NICE hierarchy is only used for propagating control messages during streaming state registration and streaming source discovery, and it is not involved in the data streaming process once the streaming source is located.

3.3 Handling Failures

The NICE hierarchy is adaptive to node failures or network partitions through automatic reconfiguration. Similarly, MetaStream handles streaming session failures due to the above reasons by relocating the streaming source, using the same content discovery protocol described above. If a failed client is able to continue relaying the stream from its cache, it can sustain the streaming until the cache is depleted, while those downstream clients re-locate alternative sources. Even if the failed client stops relaying immediately, it is unlikely to disrupt

the entire streaming of downstream clients if multiple-stripe encoding is used; they would only experience temporary deterioration of the streaming quality.

3.4 Refining and Adapting the Streaming Overlay

The streaming overlay is dynamic as clients join and depart. This raises opportunities for optimizing streaming paths. If a stripe is currently provided by the server, the server load will be reduced if the stripe can be switched to another client. In addition, the streaming efficiency of the overlay can also be improved by attaching a stripe to a closer available streaming source. Both situations can be exploited by periodically attempting to re-attach selected stripes in a controlled manner, e.g., by querying an appropriate number of levels up the content discovery hierarchy.

So far MetaStream has only considered bandwidth constraints at the edge of the network, i.e., at the server and clients. In practice, internal links in the underlying network might become the bottleneck. MetaStream can be modified to take the available bandwidth of the streaming paths into consideration. For instance, the client can select the top few closest candidates during the content discovery, measure the throughput to each of them, and choose the one with the highest throughput. During the streaming, the network may not be able to deliver the throughput required at the client. The client can adapt by dropping the current connection and re-attaching to an alternative streaming source.

4 Experimental Methodology

In this section, we describe the experimental methodology for evaluating MetaStream.

4.1 MetaStream Implementation

We have implemented MetaStream as a stand-alone application in C++, on top of the NICE code (http://www.cs.umd.edu/ suman/research/myns/index.html). The NICE code exports an API that can be used to build applications on top of the NICE hierarchy. We also modified the core API to allow for socket communication. This modification is transparent to applications built on top of the core API. In total, our implementation of MetaStream consists of about 5500 lines of code.

4.2 ModelNet Emulator

We evaluated our MetaStream implementation in the ModelNet [24] IP emulation framework. Our ModelNet setup consisted of four machines, each with a 3.0GHz Pentium IV processor and 512MB RAM, interconnected with a Gigabit Ethernet switch. Two of these machines ran Free BSD 4.7, and served as the ModelNet core cluster, while the remaining two ran RedHat 9.0 and served to support the virtual nodes. A network topology generated by GT-ITM described below was emulated by the core routers and used to connect 1000 virtual nodes.

4.3 Network Model

The experiments are conducted using a network topology with 5050 routers generated by the GT-ITM [25] graph generator using the transit-stub model. The streaming server and 1000 clients are randomly attached to the stub routers.

To model the heterogeneity of the capacities of client nodes in the Internet, we categorize clients into 3 groups according to their bandwidth similarly as in [6]: (1) 50% clients have maximum total bandwidth of 128Kbps and they represent Modem/ISDN users; (2) 35%

clients have maximum total bandwidth of 1Mbps and they have Cable Modem/DSL connections; (3) the rest 15% clients have maximum total bandwidth of 10Mbps and they have Ethernet connections. The inbound and outbound bandwidth are allocated from the total available bandwidth. We control the allocation using outbound/inbound bandwidth ratio r. The larger the ratio, the more bandwidth is contributed to streaming to other clients.

4.4 Data Model

The media object is encoded into 50 stripes using layered encoding, similarly as in [6]. All stripes are assumed to have an identical rate of 20Kbps, although in practice stripes can be encoded using different streaming rates. The total streaming rate is 1Mbps. The full length of the stream is 20 minutes (1200 seconds) and each client caches 100 seconds of received stripes after playback.

Streaming clients arrive according to the Poisson process and the average inter-arrival time is 10 seconds. Each client leaves after finishing the 20-minute streaming, and the simulation stops when the last client finishes streaming.

4.5 Protocols Evaluated

MetaStream considers both the server load and the network link cost. These two goals can conflict with each other. We are interested in whether the server load has to be compromised to achieve lower link cost. The problem of optimizing server load alone for layer-encoded on-demand streaming is NP-complete under client output bandwidth constraints [6]. In [6], a greedy algorithm is proposed and it can maximize the number of stripes that a new client can get from other clients at the moment of joining the streaming overlay. We term this greedy algorithm "Greedy:ServerLoad" and it represents the best effort towards minimizing server load to our knowledge. We implemented the "Greedy:ServerLoad" algorithm to run on the same emulator.

For comparison, we have also implemented a unicast-based on-demand streaming approach, in which each client simply streams all desired stripes from the server. Although this approach has obvious drawbacks such as overloading the server and the network links close to the server, it is widely used in the Internet today.

The NICE hierarchy used by MetaStream is maintained by *HeartBeat* messages within clusters and the default heartbeat interval is set to 3 seconds. We also set the interval for refreshing streaming buffer state to 3 seconds for both MetaStream and the "Greedy:ServerLoad" scheme.

4.6 Performance Metrics

We compare the performance of various solutions using the following metrics: (1) *Path length* measures the average number of network-layer routing hops of the streaming paths for all stripes received at any client. (2) *Link stress* measures the data traffic at the network links, averaged over all the links that have through data traffic. Together, link stress and path length give a complete picture of the network link cost of streaming. (3) *Server load* measures the bandwidth consumption at the streaming server. It reflects the effectiveness of the protocol in locating client-side streaming sources. (4) *Discovery delay* refers to the latency for a client to locate/relocate the streaming sources. (5) *Control overhead* comes from two sources in MetaStream: the overhead in constructing and maintaining the topology-aware hierarchy, and the overhead in updating and querying streaming information. All the control messages exchanged are counted in measuring the control overhead.

5 Experiment Results

We present the results from extensive experiments in this section. We first evaluate MetaStream in a typical setting and then study the effects of various system parameters.

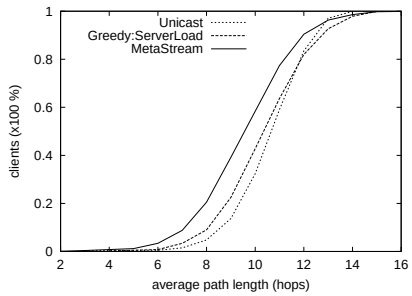

Fig. 5. Representative scenario: cumulative distribution of path length

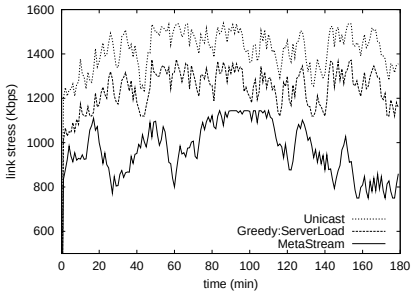

Fig. 6. Representative scenario: link stress

5.1 Representative Scenario

As a representative scenario, the outbound/inbound bandwidth ratio is set to 1.0. This models the peer-to-peer spirit which requires each node to contribute as much output bandwidth as its input bandwidth consumption.

Path Length Figure 5 shows the cumulative distribution of the average path length of all the layers received by each client. The diameter of the network topology is 20 hops. In MetaStream, on average all the layers received at each client travel at most 10 hops (half of the network diameter) for about 58.3% of the clients, but for only about 42.8% clients using the "Greedy:ServerLoad" algorithm. In addition, for each layer, MetaStream reduces the average streaming path length by more than one hop, compared with the "Greedy:ServerLoad" algorithm. result is not shown due to page limitation).

Link Stress Shorter path length translates into potentially lower link stress on the underlying network, and this is confirmed in Figure 6, which shows the average streaming traffic over network links during the simulation. MetaStream induces about 10-20% less link stress compared with the "Greedy:ServerLoad" algorithm. The unicast approach generates much higher link stress than both the "Greedy:ServerLoad" algorithm and MetaStream, which can be explained by the high bandwidth consumption near the streaming server. The results on the streaming path length and network link stress show that by taking network topology into account, MetaStream can effectively reduce the streaming cost on the backbone network.

Server Load Figure 7 shows the server load measured by the number of layers that the server has to stream. First, the "cache and relay" approach can significantly reduce the bandwidth demand at the streaming server. By using "cache and relay", MetaStream and the "Greedy:Server Load" algorithm bring server load down to below 1% of the total bandwidth demand by clients. The peak bandwidth consumption is only about 4Mbps (200 layers) at the server, which is equivalent to 4 clients with full quality streams. Even a desktop computer with Ethernet connection can sustain the server load in this scenario. Second, MetaStream achieves almost the same amount of server load as the "Greedy:ServerLoad" algorithm, which is originally designed for the purpose of minimizing server side bandwidth. This implies that MetaStream does not sacrifice server load for lower network cost and the "cache and relay" approach itself can effectively limit the server-side load. We have run both algorithms under various combinations of parameters and in all the situations that we have investigated, MetaStream achieves comparable server bandwidth savings as the "Greedy:ServerLoad" algorithm.

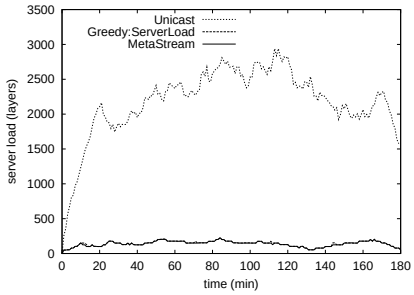

Fig. 7. Representative scenario: server load

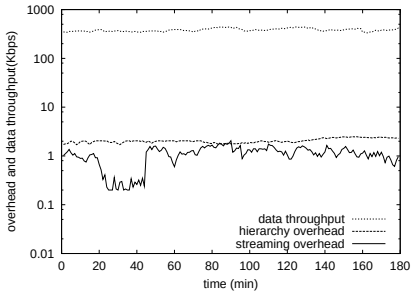

Fig. 8. Representative scenario: overhead at clients in MetaStream

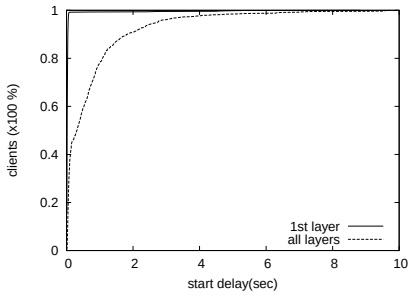

Fig. 9. Representative scenario: content discovery delay at clients in MetaStream

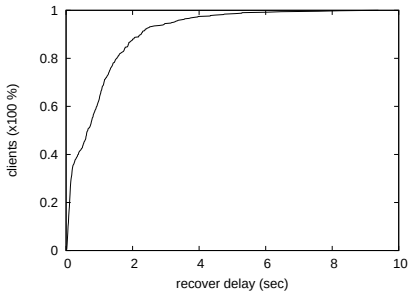

Fig. 10. Failure recover delay at clients in MetaStream

Discovery Delay Figure 9 shows the CDF of the startup delay at the clients. The link delay generated by GT-ITM is used in calculating the delay in resolving requests for stripes. Over 99% of clients experience less than 1 second delay in locating the first stripe and over 90% experience less than 2 seconds delay in locating all the stripes. The maximum delay is under 10 seconds for both data curves.

Control Overhead The control overhead of MetaStream at clients is depicted in Figure 8. The overhead bandwidth accounts for both the incoming and outgoing control traffic at clients. The volume of control traffic from streaming related activities and from the NICE protocol are comparable. The control overhead induced by the NICE protocol remains stable as clients join and leave the streaming overlay. The overhead from MetaStream itself fluctuates slightly as the number of active streaming layers changes over time (as shown in Figure 7). Both overhead are two orders of magnitude less than the streaming data received at clients.

5.2 Effects of System Parameters

The performance of MetaStream varies with the following parameters:

- *Cache size:* Long buffers imply that more streaming requests can be satisfied by other clients (Figure 11). It also allows for more choices in selecting streaming sources and therefore better quality for streaming paths (Figure 12).
- *Client output bandwidth:* The availability of a client as a streaming source is determined not only by the timing dependency between streaming requests, but also by the output bandwidth of the client. With all the other factors fixed, more abundant output bandwidth enables a client to serve more other clients (Figure 13). This can also result in shorter streaming paths for some clients (Figure 14).

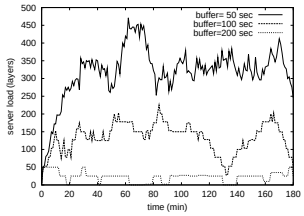

Fig. 11. Server load with varying buffer length: outbound/inbound = 1.0, average arrival interval = 10 sec

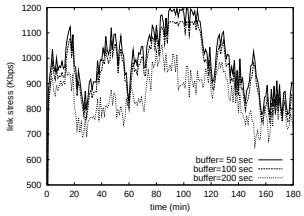

Fig. 12. Link stress with varying buffer length: outbound/inbound = 1.0, average arrival interval = 10 sec

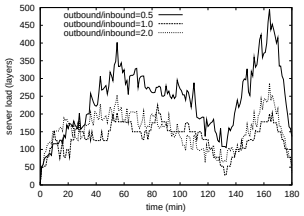

Fig. 13. Server load with varying outbound/inbound ratio: buffer length = 100 sec, average arrival interval = 10 sec

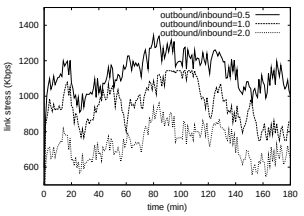

Fig. 14. Link stress with varying outbound/inbound ratio: buffer length = 100 sec, average arrival interval = 10 sec

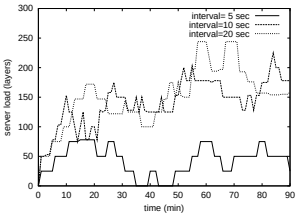

Fig. 15. Server load with varying client arrival rate: buffer length = 100 sec, outbound/inbound = 1.0

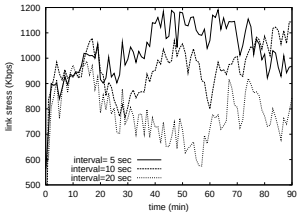

Fig. 16. Link stress with varying client arrival rate: buffer length = 100 sec, outbound/inbound = 1.0

5.3 Scalability

A third parameter that can affect the performance of MetaStream is the client arrival rate. For a fixed streaming length, higher arrival rate translates into larger client population, and thus varying the arrival rate measures the scalability of MetaStream. On one hand, increased client population increases the aggregate bandwidth demand. On the other hand, it increases the opportunity for "cache and reply" from other clients instead of from the server. Figure 16 and Figure 15 show that MetaStream scales well with the client population, in terms of the bandwidth demand on both the server side and the underlying network.

5.4 Fault Recovery

In order to evaluate the failure recovery capability of MetaStream, we introduced random failures in our experiment of the representative scenario. Out of the 1000 clients, 30% fail before they finish the 20 minute streaming. Failures are randomly injected among the clients and for each failed client, the failure time is determined by a random value in between (1, 20) minutes.

We also assume that clients fail silently and stop streaming to all downstream clients immediately. Since the stream is encoded using the layer encoding scheme, once a layer is interrupted due to source failure, all higher layers become useless and are therefore also disconnected from their corresponding streaming sources. All disrupted layers will be recovered by the failure handling mechanism. The directly affected clients continue to stream to their own downstream receivers, and thus their caches can absorb some of the potentially cascading effects of upstream failures in the streaming overlay.

Figure 17 and Figure 18 show the streaming throughput received and the streaming quality experienced by an average client, with and without failure recovery, respectively. Client

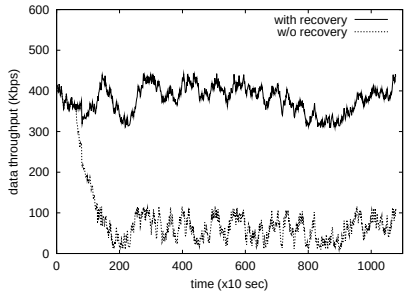

Fig. 17. Throughput at clients, with failures

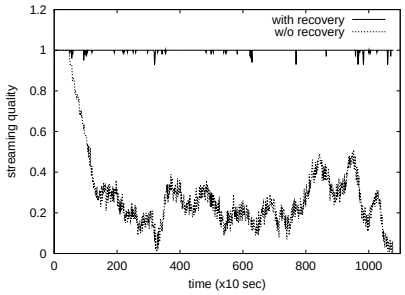

Fig. 18. Streaming quality at clients, with failures

streaming quality is defined by the percentage of decodable layers out of all the desired layers. These results show that MetaStream can fully restore all the streaming sessions and clients are hardly affected by failures except a tiny delay for recovery. Meanwhile the server load remains roughly at similar levels as in failure-free situations (the result is omitted due to page limit). Figure 10 depicts the CDF of the delay to recover from streaming source failures, i.e., the time from when the streaming quality at the client declines due to source node failures to when the streaming quality fully recovers. The failure recovery delay is comparable with the startup delay shown in Figure 9.

6 Related Work

MetaStream is related to previous work in the area of on-demand streaming. There has been a rich body of work on on-demand streaming which roughly fall into two categories: multicast-based techniques [12, 15, 9, 8] and media caching [20, 3, 23, 4]. MetaStream is closely related to several recent work on client-side media caching [5, 6, 14]. Also related to our work is overlay-based multicast and streaming [13, 18, 1, 17, 2, 19, 16, 11, 22]. Compared with these previous work, MetaStream focuses on the problem of content discovery in peer-to-peer "caching and relaying", and addresses the server load, network cost, and client heterogeneity issues in a practical and efficient framework.

7 Conclusions

This paper presents the design and implementation of MetaStream, a scalable and distributed content discovery protocol for high bandwidth and low network cost media streaming in a cooperative environment. Specifically, this paper makes the following contributions:

- We presented the design and evaluation of MetaStream, which creates a media distribution network based on a topology-aware overlay and allows streaming clients to achieve high bandwidth in caching-and-relaying on-demand media while incurring low link cost to the underlying network;
- We conducted a large-scale evaluation of 1000 streaming clients running in an emulated 5050-router network topology. The experiment results show that MetaStream incurs low network link stress and low server load by effectively locating nearby client-side streaming sources, and the streaming overlay is highly resilient to overlay membership changes and client failures.

Acknowledgment

This work was supported in part by an NSF CAREER award (ACI-0238379).

References

1. S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM*, August 2002.
2. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwidth Content Distribution in Cooperative Environments. In *Proceedings of ACM SOSP*, October 2003.
3. Y. Chae, K. Guo, M. M. Buddhikot, S. Suri, and E. Zegura. Silo, Tokens, and Rainbow: Schemes for Fault Tolerant Stream Caching. *IEEE JSAC, Special Issue on Internet Proxies*, April 2002.
4. S. Chen, B. Shen, S. Wee, and X. Zhang. Adaptive and Lazy Segmentation Based Proxy Caching for Streaming Media Delivery. In *Proceedings of ACM NOSSDAV*, June 2003.
5. Y. Cui, B. Li, and K. Nahrstedt. oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks. *IEEE JSAC, Special Issue on Recent Advances in Service Overlays*, 2004.
6. Y. Cui and K. Nahrstedt. Layered Peer-to-Peer Streaming. In *Proceedings of ACM NOSSDAV*, June 2003.
7. A. Dan and D. Sitaram. A Generalized Interval Caching Policy for Mixed Interactive and Long Video Workloads. In *Proceedings of SPIE MMCN*, January 1996.
8. D. Eager, M. Vernon, and J. Zahorjan. Minimizing Bandwidth Requirements for On-Demand Data Delivery. *IEEE Transactions on Knowledge and Data Engineering*, 13(5), 2001.
9. L. Gao and D. Towsley. Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast. In *Proceedings of IEEE ICMCS*, 1999.
10. V. K. Goyal. Multiple Description Coding: Compression Meets the Network. *IEEE Signal Processing Magazine*, 2001.
11. M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. Promise: Peer-to-Peer Media Streaming Using Collectcast. In *Proceedings of ACM Multimedia*, November 2003.
12. K. Hua. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems. In *Proceedings of ACM SIGCOMM*, September 1997.
13. Y. hua Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.
14. S. Jin and A. Bestavros. Cache-and-Relay Streaming Media Delivery for Asynchronous Clients. In *Proceedings of NGC*, October 2002.
15. S. S. Kien A. Hua, Ying Cai. Patching: A Multicast Technique for True Video-on-Demand Services. In *Proceedings of ACM Multimedia*, September 1998.
16. D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of ACM SOSP*, October 2003.
17. Z. Li and P. Mohapatra. HostCast: A New Overlay Multicasting Protocol. In *Proceedings of IEEE ICC*, May 2003.
18. J. Liebeherr, M. Nahas, and W. Si. Application-Layer Multicasting with Delaunay Triangulation Overlays. In *Proceedings of IEEE GLOBALCOM*, May 2001.
19. V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient Peer-to-Peer Streaming. In *Proceedings of IEEE ICNP*, November 2003.
20. S. Ramesh, I. Rhee, and K. Guo. Multicast with Cache (Mcache): An Adaptive Zero-Delay Video-on-Demand Service. In *Proceedings of IEEE INFOCOM*, April 2001.
21. R. Rejaie, M. Handley, and D. Estrin. Layered Quality Adaptation for Internet Video Streaming. *IEEE JSAC, Special Issue on Internet QOS*, 2000.
22. R. Rejaie and A. Ortega. PALS: Peer-to-Peer Adaptive Layered Streaming. In *Proceedings of ACM NOSSDAV*, June 2003.
23. S. Sheu, K. A. Hua, and W. Tavanapong. Chaining: A Generalized Batching Technique for Video-On-Demand Systems. In *Proceedings of IEEE ICMCS*, June 1997.
24. A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proceedings of USENIX OSDI*, December 2002.
25. E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE INFOCOM*, March 1996.