Enabling DVD-like Features in P2P Video-on-demand Systems

Nevena Vratonjić
School of Computer and Communication Sciences, EPFL
Lausanne, Switzerland
nevena.vratonjic@epfl.ch

Priya Gupta
Computer Science and Engineering Department, IIT Delhi, India
cs1040177@cse.iitd.ernet.in

Nikola Knežević
School of Computer and Communication Sciences, EPFL
Lausanne, Switzerland
nikola.knezevic@epfl.ch

Dejan Kostić
School of Computer and Communication Sciences, EPFL
Lausanne, Switzerland
dejan.kostic@epfl.ch

Antony Rowstron
Microsoft Research
Cambridge, UK
antr@microsoft.com

ABSTRACT

Peer-to-peer (p2p) video-on-demand (VoD) is increasingly popular with Internet users. Currently deployed pure p2p VoD systems provide poor general performance and they lack advanced features such as fast forward and seeking to arbitrary points. Peer-assisted VoD systems can provide such services, but they require very well provisioned source servers (or server farms).

We propose BulletMedia, a system that uses proactive caching to attempt to provide advanced features without requiring a well provisioned server. In BulletMedia, blocks are altruistically replicated by peers not to aid immediate playback but to simply increase the number of replicas of each block. This helps ensure that blocks are available in-overlay and reduces dependence on the source. BulletMedia combines a traditional overlay mesh approach with a structured overlay. The overlay mesh is used to fetch blocks at a high rate, while the structured overlay is used to enable efficient block discovery and to control block replication. Initial experimental results from a prototype BulletMedia implementation demonstrate that it can both effectively control in-overlay block replication and can efficiently use these replicas to perform forward seeks.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; H.4.3 [**Information Systems Applications**]: Communications Applications

General Terms

Experimentation, Performance

Keywords

Video-on-demand, Overlays, Peer-to-peer

1. INTRODUCTION

Peer-to-peer (p2p) video-on-demand (VoD) services are becoming increasingly popular. These services have the potential to revolutionize how people view media content. However, the current VoD services offer a different playback experience from download-based systems, like BitTorrent. Download-based systems require the content to be downloaded in its entirety, but once downloaded provide the freedom to fast-forward, rewind and seek to arbitrary locations in the content during playback. P2p VoD systems do not support this kind of interactive playback: they depend on being able to spend minutes caching content before commencing playback and then rely on contiguous playback from the beginning of the media to the end.

To build p2p VoD systems that support these advanced operations, like fast-forward and seeking to arbitrary play points, is challenging. When a peer changes its play point the peer needs to rapidly discover other peers that have replicated the content blocks it requires. If it cannot find the blocks, they need to be fetched from the source server. In general, in a fully-centralized approach, if there are N users then this requires $O(N)$ bandwidth at the server. In a poorly designed p2p VoD systems, where peers can perform random seeks to any playback point, then the required bandwidth at the source can still be $O(N)$ under certain conditions.

The main contribution of this work is the initial design and prototype implementation of a p2p VoD system with DVD-like features called BulletMedia. As with other p2p VoD systems the content being distributed is decomposed into a set of equal size blocks. The basic concept behind BulletMedia is to ensure that *all* blocks are replicated in-overlay, regardless of when the set of active peers in the overlay will require them to support current playback. This relies on peers altruistically fetching and storing blocks that they do not require for immediate playback. Increasing the availability of blocks in-overlay reduces dependency on the source server and reduces the latency of resuming playback when a peer performs a seek operation.

Key challenges for BulletMedia are controlling which block should be replicated, when peers have spare bandwidth, and supporting efficient block discovery. If a block is not replicated or cannot be discovered then the peer is forced to fetch it from the source server. BulletMedia uses *proactive caching* to achieve this and leverages both a mesh and structured overlay. The mesh overlay is used to fetch blocks under normal playback. The structured overlay is used to control both which blocks are altruistically replicated, based on the current number of replicas of a block and to allow peers to rapidly discover the location of blocks stored in-overlay. In order to demonstrate the feasibility of our approach we present some initial results gathered using a prototype implementation of BulletMedia which demonstrates the effectiveness of BulletMedia at both controlling block replication and at supporting forward seek operations during playback.

The rest of this paper is organized as follows. The next section describes the design of BulletMedia. In Section 3 we present our experimental results. Section 4 overviews the related work, while Section 5 concludes the paper and outlines future work.

2. DESIGN

We assume that initially there is a source server that contains the entire media file, decomposed into blocks. End-users run an instance of a p2p media player on their endsystems and specify the media file they would like to play. To join the overlay that peer can either contact the source server or contact any other arbitrary peer in the system. Further, we assume that peers can join and fail at any time. After as small a delay as possible the media playback commences. Once playback has commenced the client supports advanced operations like random seeks, fast-forwarding and rewinding.

Next, we describe the main BulletMedia components: the high-bandwidth overlay mesh and the structured overlay used to enable proactive caching and advanced operations.

2.1 Overlay mesh

The overlay mesh is built on top of Bullet′ [10] and RanSub [11]. As shown in Figure 1, each peer is connected to a number of neighbors in the mesh. The set of neighbors sending data to a peer are *senders*, while those retrieving data from it are *receivers*. BulletMedia is *pull-based*; a peer first learns of available blocks at its senders and makes explicit requests for required blocks to achieve high throughput.

Neighbor discovery is performed using RanSub, which allows sampling of system-wide state. RanSub acts as a simple control plane, building and maintaining a spanning tree across all peers. In BulletMedia this control tree is used to distribute overlay membership information. In particular, when a peer joins BulletMedia, it first joins the RanSub control tree. The peer starts to periodically receive information about other peers selected uniformly at random from the set of all peers which are part of the overlay. As well as discovering the identity of the peers each peer also finds coarse-grain information about the blocks stored at those peers.

The information provided by RanSub allows BulletMedia peers to pick an initial set of senders on joining. Once joined BulletMedia peers can dynamically change the set of neighbors using a neighbor set sizing algorithm to attempt to maximize the received and forwarded bandwidth. In addition, slow senders and receivers getting a small fraction of bandwidth are periodically disconnected to further foster mesh adaptation to dynamic network conditions. A dynamic flow control algorithm determines the maximum amount of outstanding data requested from each sender. BulletMedia peers make entirely local decisions; e.g., each receiver tries to maximize its bandwidth utilization.

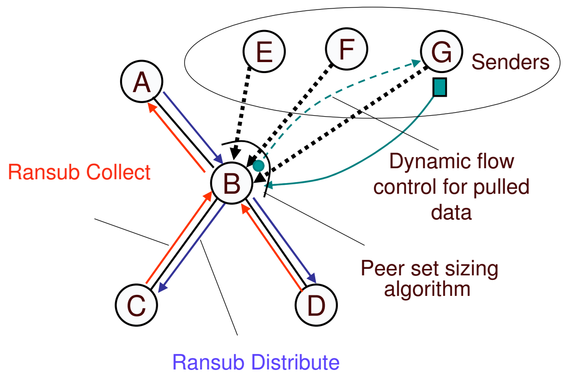
A
E
F
G
Senders
Ransub Collect
B
Dynamic flow control for pulled data
Peer set sizing algorithm
C
D
Ransub Distribute

Figure 1: Overview of overlay mesh creation in BulletMedia.

Each peer is configured with a small fixed-size ordered *playback buffer* and a *content cache*. The playback buffer is used to store blocks that are required imminently by the video codec. Blocks currently in the playback buffer as well as already used blocks and blocks that will be used in the future are stored in the content cache. The head of the playback buffer is the *play point* and is the next block required by the video codec to continue playback. Over time, the members of a peer's neighbor set will mostly have overlapping play points.

To monitor the play back quality, each peer locally measures a *continuity index* (*ci*) which captures the fraction of blocks delivered in time over a small window (e.g. 10 blocks) before the play point. If *ci* drops below a predetermined threshold then playback is paused, while more blocks are retrieved.

The BulletMedia flow control algorithm controls when blocks can be retrieved from each sender. The block to be requested is selected randomly from the set of blocks with the lowest availability across all senders; referred to as the *rarest random* strategy. This strategy has been shown to perform better than other strategies [10]. In addition, a receiver pays attention to the impending block delivery deadline to the codec, and explicitly requests a few blocks that are adjacent to the play point. Small block sizes (e.g. 16 kB) are used to allow the mesh to achieve high network bandwidth utilization.

2.2 Structured overlay

In general peers select a set of senders that provide high download rates and contain blocks required for playback which removes the need to fetch blocks from the source.

If a peer moves its play point, for example by performing a seek operation, it then needs to rapidly discover and fetch the blocks required for the playback buffer. Playback cannot commence until enough blocks have been fetched. If

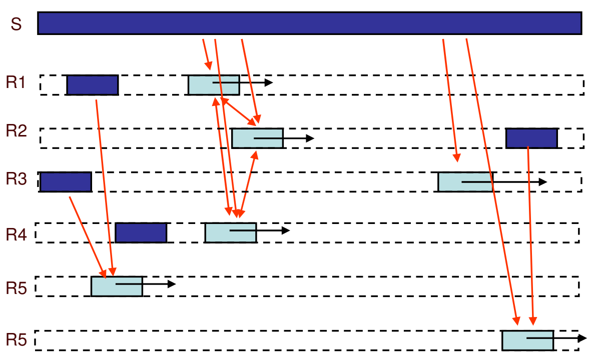

Figure 2: Caching and proactive placement of chunks in BulletMedia. ***S*** **is the source node; the playing buffers are lightly-colored, while the dark-shaded blocks represent cached content.**

the senders of the peer cannot provide the blocks, the peer fetches the blocks from the content source server (e.g., peer 3 in Figure 2). If many peers are concurrently moving their play points this can lead to high peak-bandwidth demands on the source server.

In order to address this issue BulletMedia uses a distributed content discovery service. A Distributed Hash Table (DHT) is used to store meta-data about content location within the peers of the mesh overlay. The DHT does not store meta-data at a block granularity as these are small and maintaining meta-data at this granularity incurs a high overhead. Instead, sets of contiguous blocks are deterministically grouped into *chunks* (e.g., 100 blocks in a chunk). Each chunk is assigned a unique key, *chunkId*, which is generated by hashing $< contentname, chunknumber >$. This ensures that any peer, given that it knows the number of blocks per chunk, can determine the chunkId of the chunk that contains any specific block.

Each peer monitors its local content cache and when all blocks associated with a particular chunkId are present the peer inserts an entry into the DHT. The entry has the chunkId as the key, and the peer's IP address as the value. All inserted meta-data is tagged with a time-to-live to ensure that it times out appropriately.

When a peer performs a seek operation, thereby moving the play point, it first checks the locally available sender set block maps. If it is unable to fetch the required blocks from its sender set, the peer determines the chunkId of the block and queries the DHT. The DHT returns a (bounded) set of peers associated with that chunkId. The peer randomly chooses one or more of the returned peers and adds them to the sender set.

This process enables peers to rapidly discover new peers with appropriate blocks, reducing the dependency on the source server.

As described earlier, chunks are used to ensure that the overhead of maintaining the meta-data in the DHT is reasonable. Using chunks has an additional advantage: once a peer has discovered a few peers with the chunks it needs, it is able to retrieve multiple blocks from each peer, aggregating the peering (e.g. TCP three-way handshake and slow-start) overheads.

2.3 Managing the content cache

We assume that the peers are able to cache the entire set of chunks associated with the current media being played. If peers have smaller caches then they can query the DHT to discover the number of other peers that have a particular chunk replicated. Based on the peer's local play point and the numbers of replicas of each chunk a peer can decide which blocks to delete to minimize impact on block availability.

Simply populating each peer's content cache with content required for imminent playback and blocks already played is insufficient. If a peer moves its play point beyond the maximum play points of all other participating peers then the peer will be forced to fetch content from the source server.

BulletMedia uses *proactive caching* of blocks: the aim is to ensure all content blocks are replicated at least k times system-wide, where a typical value of k is 4. This is similar in spirit to the approach taken by some P2P storage systems, e.g., TotalRecall [2].

Each peer uses its spare bandwidth to prefetch blocks in advance. These blocks are not required for imminent playback but are altruistically fetched to ensure diversity of blocks stored in the overlay.

When a peer believes that it has spare incoming bandwidth, for example because it has sufficient blocks cached to populate the playback buffer, it must determine which blocks it should prefetch in order to ensure good diversity across the overlay.

To achieve this a peer examines its local content cache and determines the set of chunkIds for chunks it is currently not replicating. It then selects chunkIds at random from this set, and performs a lookup in the DHT. The DHT simply includes a count of the number of peers replicating the chunk in the response. If the number of replicas is below a pre-defined level (e.g., 4) then the peer begins to retrieve the blocks associated with the chunk. If not, the peer chooses another chunkId at random and queries it. This process proceeds until a chunk is found that requires more replication. If no chunk is found that requires more replicas, then the peer fetches blocks based on it local playback requirements. Otherwise (the chunk is under-replicated), the peer knows the estimate of number of replicas of a chunk in the mesh and the identities of the peers storing them. If there are one or more replicas, then the peer adds these peers to the sender set. If possible, the peer avoids the source (content server) to reduce its sending load.

When a peer is altruistically fetching blocks if it determines that blocks are required to support the current play back, the peer stops fetching blocks for the selected chunk. The content server also prioritizes delivery of blocks required for imminent playback over blocks that are being fetched to increase a chunks replication factor in the overlay.

3. EVALUATION

In order to evaluate BulletMedia we have created a prototype implementation using Mace [13], a framework for implementing distributed systems. The mesh overlay is based on the Bullet' implementation in Mace and the DHT is an implementation of Pastry [14], also implemented in Mace.

The prototype implementation is currently limited to disseminating a single movie and supports seeks to arbitrary play points within the movie. We intend to add more functionally, such as fast-forward in the future.

Integral to the design of BulletMedia is the ability for peers to determine if they have spare bandwidth. Determining link capacities is difficult, and the prototype implementation uses a number of simple heuristics to determine if there is spare bandwidth. We believe that in general for VoD applications the ability for peers to determine spare link capacities will be very powerful. Beside being useful for deciding when to prefetch blocks, available bandwidth estimation can enable features like admission control. While the simple heuristics are effective, we intend to investigate this aspect further in the future. Unlike the peers, the source server manages its outbound bandwidth by statically allocating two times the playback bandwidth for providing blocks for imminent playback and using the rest for supporting altruistic prefetching of blocks.

In our evaluation, we examine 1) whether BulletMedia peers replicate the file the specified number of times, and 2) the system's ability to accommodate a large number of random seek requests without stalling the video playback.

3.1 Experimental Setup

We conducted our experiments using the ModelNet [15] network emulator. We make use of 8 dual 3.4-Ghz Pentium-4s running Linux 2.6.17 that are interconnected by a 1-Gbps Ethernet switch. We multiplex one hundred logical end peers running our applications across the 8 Linux machines (slightly over 12 instances on average per machine). ModelNet routes packets from the end systems through an emulator responsible for accurately emulating the hop-by-hop delay, bandwidth, and congestion of a given network topology; a 3.4-Ghz Pentium-4 running FreeBSD 4.9 served as the emulator for these experiments.

For all our experiments, we use a 5,000-node INET [4] topology that we further annotate with bandwidth capacities for each link. We keep the latencies generated by the topology generator; the average network RTT is 130ms. We randomly assign our participant peers to act as clients connected to one-degree stub nodes in the topology. We randomly select one of these participants to act as the video source and set its outbound bandwidth to 14 Mbps. We set all links except the client-stub (access) links to be 100 Mbps, with latency chosen by the INET topology generator. Other participants have 2 Mbps inbound/outbound access link bandwidth. We use TCP for all data and control messages.

The movie file is 50 MB with the video rate of 600 kbps. The size of the playing buffer is 100 blocks, each block being 16 kB in size. We set the chunk size to 100 blocks. As a bigger file would take longer to be replicate, varying the replication factor (k) might be useful. In our experiments we set $k = 4$, giving BulletMedia a chance to replicate the file before random seeks begin. The default minimum playing rate is set to 0.8; this means that we assume that the video can be viewed only if more than 80 percent of the blocks are delivered to the codec.

3.2 Random Seek Performance

In this section, we demonstrate the ability of our system to support random seeks under conditions that severely strain traditional systems. To subject the system to an initial flash crowd, all (hundred) peers start playing simultaneously from the beginning of the movie. To effect a severe random seek

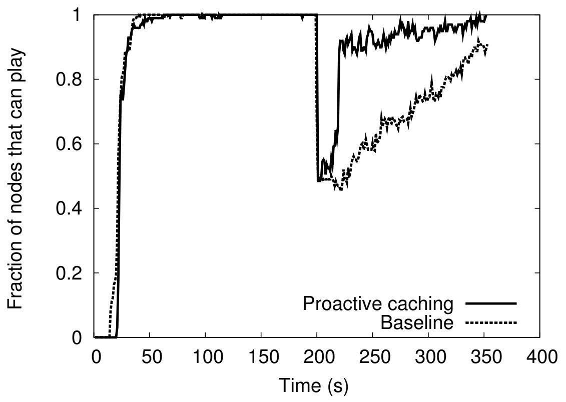

Figure 3: Performance in a random seek scenario.

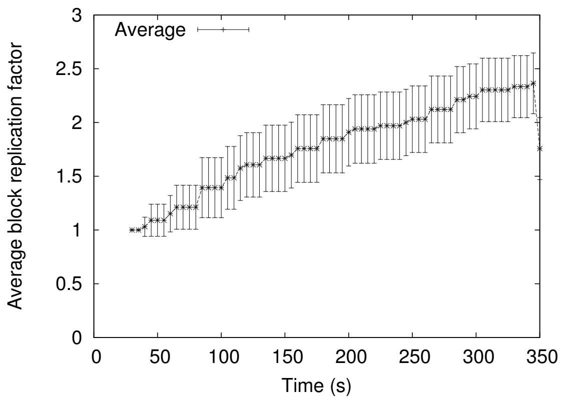

Figure 4: Average block replication factor over time (baseline)

scenario, at $t = 200$ seconds we allow 50 peers (half of the system) to perform *forward* seeks to a random part of the movie that has not been played yet and then stay in the play mode for a random amount of time. Peers performed two or three seeks and stayed in play mode between seeks randomly from 40 to 60 seconds. Our experiments last 350 seconds.

Figure 3 shows the fraction of peers that can play (ideally 1.0) over time. Our reference point is the baseline system, namely BulletMedia that does not employ proactive caching and instead only lets receivers advertise content they have played out. The second line is for BulletMedia with proactive caching. We see that both systems take about 20 seconds to cache the initial blocks required for the codec and then have very little difficulty in accommodating all receivers. The source is pushing data at 1.2 Mbps and has 10 Mbps used for serving individual block requests. Once random seeks begin, the peers in the baseline system have great difficulty locating and downloading content as the source has only 10 slots for downloaders. There is some overlap between random seek points though, so peers gradually self-organize over time to improve each other's performance. With proactive caching however, more than 90 percent of receivers quickly start playing after the random seek. The seek latency resembles the initial join time, where approximately hundred blocks have to be cached for movie play

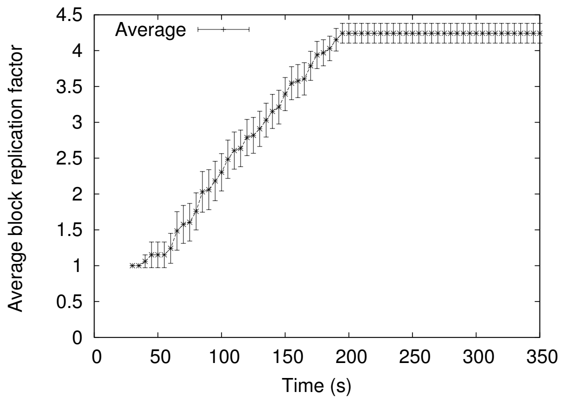

Figure 5: Average block replication factor over time (Proactive caching, target replication factor is 4).

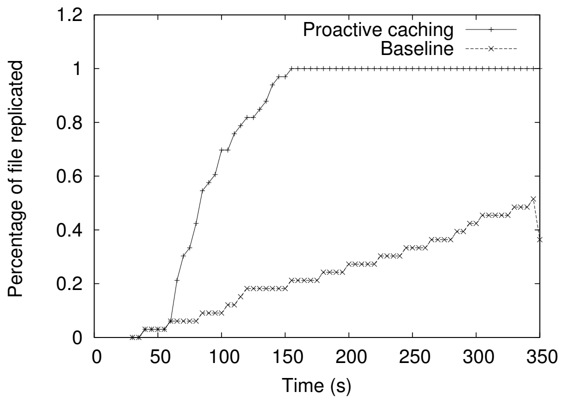

Figure 6: Percentage of file covered by replication over time.

to begin. This figure clearly shows the benefits of proactive caching and the ability of our system to quickly locate and retrieve blocks. The fraction of peers that can play after seeks is not at the maximum due to a smaller number of viable senders (on average each chunk is replicated 3 times at non-source peers); downloading using TCP from just a few senders over a wide-area topology results in occasional stalls. In addition, each sender has just about enough outbound bandwidth to support its mesh-based receivers and the downloads that are redirected to it via the DHT (three, on average).

To shed more light on the observed performance, we plot the average block replication factor for the baseline and BulletMedia cases in Figures 4 and 5, respectively. We also show the fraction of the file that is replicated system-wide in Figure 6. These figures show that the baseline system does not manage to replicate the entire file before the seeks begin (it does not even attempt to). On the other hand, BulletMedia replicates each block steadily, with low standard deviation (shown as error bars in Figure 5) and quickly reaches the desired replication factor of 4 (including the file present at the source, which why is the line in Figure 5 starts at 1). Even though peers make random replication choices, 80 percent of the file is replicated slightly after hundred seconds, with the entire file being covered at 150 seconds.

4. RELATED WORK

There has been considerable interest in overlay-based media distribution including live-streaming [3, 7, 9] and file distribution [6, 10, 12].

Recently, most interest has been in overlay-based VoD (or near-VoD) using unstructured overlays, for example CoolStream [16], GridCast [5], RedCarpet [1]. Most of these modify the block fetching policies used in file distribution systems to enable contiguous playback of content. BulletMedia differs from these approaches in that it actively attempts to replicate *all* blocks from the media file in the overlay: it provides mechanisms (using a structured overlay) to control which blocks are replicated and provides an efficient technique for discovering these blocks. Subsequently, BulletMedia can more efficiently support advanced operations like seeking.

Peer-assisted VoD [8] attempts to provide VoD functionality by relying on the source server (or server farm) having sufficient bandwidth to essentially support all users, and then where possible to use P2P techniques to reduce the server load. Peer-assisted VoD can support seek functionality, but requires significant bandwidth resources at the server. BulletMedia attempts to provide the functionality with lower server bandwidth overheads.

A recent study of a deployed large-scale VoD system [5] shows, when access is provided to a large number films, a heavy-tail effect for content popularity. The implication is that the system needs to support a large number of concurrent streams, but with most having a small number of active subscribers. In future work we intend to examine how easy it is to extend BulletMedia to manage multiple media files concurrently.

5. CONCLUSION AND FUTURE WORK

P2p VoD systems that provide advanced functionality, like fast-forwarding and seeking, can lead to high potential peak bandwidth at the source when peers change play points. In this paper we advocate using proactive in-overlay caching, utilizing spare bandwidth to increase per-block replication within the overlay. To achieve this BulletMedia uses both a mesh overlay and a structured overlay. The mesh overlay is used to fetch blocks under normal operation. The structured overlay is leveraged to provide both rapid discovery of peers with required blocks when peers perform seeks and to control which blocks peers should replicate. Initial results demonstrate that this is a very promising approach.

Future work focuses on improving the performance of BulletMedia, and in particular bandwidth management. As part of this we are investigating admission control: allowing peers to only join the overlay when there is sufficient bandwidth available to ensure that the current members of the overlay will not experience any negative impact.

6. REFERENCES

[1] Siddhartha Annapureddy, Christos Gkantsidis, Pablo Rodriguez Rodriguez, and Laurent Massoulie. Providing Video-on-Demand using Peer-to-Peer Networks. In *Proceedings of IEEE Infocom*, 2007.

[2] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total Recall: System Support for Automated Availability Management. In *Proceedings of the 1st USENIX/ACM*

Symposium on Networked Systems Design and Implementation (NSDI), March 2004.

[3] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth Content Distribution in Cooperative Environments. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, October 2003.

[4] Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott Shenker, and Walter Willinger. Towards Capturing Representative AS-Level Internet Topologies. In *Proceedings of ACM SIGMETRICS*, June 2002.

[5] Bin Cheng, Xuezheng Liu, Zheng Zhang, and Hai Jin. A Measurement Study of a Peer-to-Peer Video-on-Demand System. In *Proceedings of the International Peer to Peer Symposium (IPTPS)*, February 2007.

[6] Bram Cohen. Incentives Build Robustness in BitTorrent. In *Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[7] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. In *Proceedings of ACM SIGCOMM*, August 2001.

[8] Cheng Huang, Jin Li, and Keith W. Ross. Peer-Assisted VoD: Making Internet Video Distribution Cheap. In *Proceedings of the International Peer to Peer Symposium (IPTPS)*, February 2007.

[9] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and Jr. James W. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of Operating Systems Design and Implementation (OSDI)*, October 2000.

[10] Dejan Kostić, Ryan Braud, Charles Killian, Erik Vandekieft, James W. Anderson, Alex C. Snoeren, and Amin Vahdat. Maintaining High Bandwidth under Dynamic Network Conditions. In *Proceedings of the USENIX 2005 Annual Technical Conference*, April 2005.

[11] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, Abhijeet Bhirud, and Amin Vahdat. Using Random Subsets to Build Scalable Network Services. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, March 2003.

[12] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, October 2003.

[13] Mace: a toolkit and language for building distributed systems. http://mace.ucsd.edu.

[14] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems*, 2001.

[15] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[16] Xinyan Zhang, Jiangchuan Liuy, Bo Liz, and Tak-Shing Peter Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In *Proceedings of IEEE INFOCOM*, 2005.