AnySee: Peer-to-Peer Live Streaming

Xiaofei Liao, Hai Jin, *Yunhao Liu, *Lionel M. Ni, and Dafu Deng

School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, 430074, China
{xfliao, hjin, dfdeng }@hust.edu.cn

*Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{liu, ni}@cs.ust.hk

***Abstract* — Efficient and scalable live-streaming overlay construction has become a hot topic recently. In order to improve the performance metrics, such as startup delay, source-to-end delay, and playback continuity, most previous studies focused on intra-overlay optimization. Such approaches have drawbacks including low resource utilization, high startup and source-to-end delay, and unreasonable resource assignment in global P2P networks. Anysee is a peer-to-peer live streaming system and adopts an inter-overlay optimization scheme, in which resources can join multiple overlays, so as to (1) improve global resource utilization and distribute traffic to all physical links evenly; (2) assign resources based on their locality and delay; (3) guarantee streaming service quality by using the nearest peers, even when such peers might belong to different overlays; and (4) balance the load among the group members. We compare the performance of our design with existing approaches based on comprehensive trace driven simulations. Results show that AnySee outperforms previous schemes in resource utilization and the QoS of streaming services. AnySee has been implemented as an Internet based live streaming system, and was successfully released in the summer of 2004 in CERNET of China. Over 60,000 users enjoy massive entertainment programs, including TV programs, movies, and academic conferences. Statistics prove that this design is scalable and robust, and we believe that the wide deployment of AnySee will soon benefit many more Internet users.**

Keywords* — *Peer-to-Peer; Live Streaming; Inter-Overlay Optimization; Distributed Approach; Load Balance; AnySee

I. Introduction

With the improvement of network bandwidth, multimedia services based on streaming live media, such as IPTV [5], have gained much attention recently. Significant progress has been made on the efficient distribution of live streams in a real-time manner over a large population of spectators with good QoS [4]. Due to the practical issues of routers, IP multicast [6] has not been widely deployed. Therefore, researchers have expended a lot of effort building an efficient streaming overlay multicast scheme based on P2P networks [7], in which spectators behave as routers for other users. Efficient and scalable live-streaming overlay construction [8] has become a hot topic. Different from traditional distributed systems, streaming overlays focus on the following three metrics: startup delay, source-to-end delay, and playback continuity, as these metrics have a direct bearing on the interactive usability of a live streaming system. Large delays would exhaust user patience and unplanned interruptions would spoil the entertainment value.

In order to improve the above metrics, previous studies [9] focused on intra-overlay optimization, in which each node joins at most one overlay. With the help of locality-aware strategies [10][11] and optimization schemes such as DONet in CoolStreaming [12], Narada in ESM [13], QoS of live streaming P2Ps have significantly improved. However, they still suffer from long delay and unplanned interruptions, especially when a large number of peers join the network simultaneously.

Figure 1 shows an example of intra-overlay optimization with two logical streaming overlays. Peers A, B, C and D join the stream originating at S_1 and peers E, F, G, H and K join the stream originating at S_2. The number on each edge represents the cost of the link between two nodes. In traditional intra-overlay optimization schemes, two multicast trees can be established as shown in Fig. 1 (a) and (b). There are two obvious drawbacks. First, such overlay construction is not globally optimal. Considering peer D in Fig. 1(a), the cost $S_1 \rightarrow$ D is 8, while if the path $S_1 \rightarrow S_2 \rightarrow$ D is used, the cost is only 4. Second, resource utilization of traditional approaches is relatively low. Most of the existing protocols are tree based. Consequently, all leaf nodes fail to contribute any bandwidth or CPU cycles to the multicast trees.

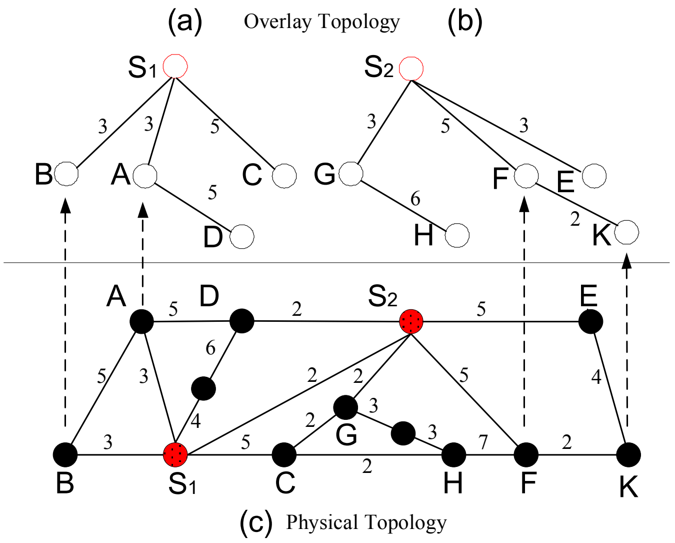


Figure 1. Intra-overlay optimization: (a) optimal multicast tree rooted at S_1; (b) optimal multicast tree rooted at S_2; (c) physical topology

We propose an inter-overlay optimization based scheme, AnySee, in which resources can join multiple overlays simultaneously, so as to (1) improve global resource utilization of a P2P live streaming network and distribute traffic to all physical links evenly; (2) assign resources based on their locality and delay; (3) guarantee streaming service quality by using the nearest peers, even if such peers might belong to different overlays; and (4) balance the load among the group members. After AnySee optimization on the example shown in Fig. 1, better overlays are constructed as illustrated in Fig. 2.

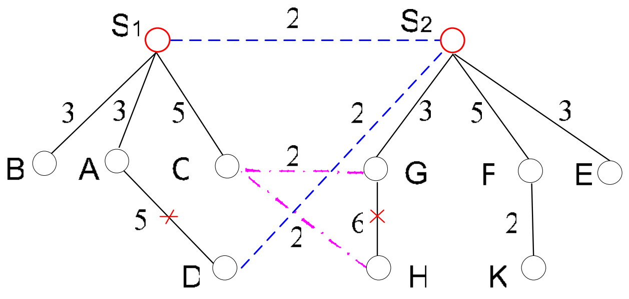


Figure 2. AnySee inter-overlay optimization

However, for a distributed approach such as AnySee, to reach the above design goal without global network knowledge is not trivial. Several key issues, including efficient neighbor discovery, resource assignment, overlay construction and optimization, must be addressed.

To prove the effectiveness of AnySee, comprehensive trace driven simulations are conducted based on topologies from real P2P networks [1]. Results show that AnySee outperforms previous schemes in resource utilization and the QoS of streaming services. A well-known public implementation, AnySee v.1.1, was released on June 2004. It has been used to broadcast live-streaming media, including TV programs, movies and the Grid and Cooperative Computing (GCC'04) international conference in Wuhan, to tens of thousands of end-users in CERNET (China Education and Research Network). In the past several months, over 60,000 users, from 40 universities and 20 cities in China, have tested AnySee P2P streaming services. The source-to-end delay, resource utilization, and the startup delay were all quite encouraging.

The rest of this paper is organized as follows. Section II discusses the related work. Section III presents the idea of inter-overlay optimization of AnySee. Section IV describes our simulation methodology and performance analysis. We describe implementation experiences and show our observations and measurements of AnySee in Section V. We conclude this work in Section VI.

II. RELATED WORKS

Two types of schemes based on intra-overlay optimization were proposed recently: tree-based overlays and mesh-based overlays. Borrowing ideas from IP multicast, tree-based protocols are simple, efficient, and scalable. There are two types of tree-based protocols, including single tree protocols, such as ESM, NICE [19] and ZigZag [18], and multiple tree protocols [14][15]. The major issue of single tree protocols is to build a scalable multicast tree with high efficiency. Multiple tree protocols, such as MDC [16], emphasize the overall resilience and load balance of the streaming network. The main idea is to divide the video of one stream into several parts based on "layer concept" in CoopNet or patching ideas [20]. However, the leaving or crash behavior of nodes in the upper layers often causes buffer underflow. They cannot provide backup streaming services, and waste any spare resources.

To improve the stability of services, mesh-based protocols have been proposed, in which each peer can accept media data from multiple "parents" as well as providing services for multiple "children", such as Coolstreaming, PROMISE [17] and GNUStream [21]. The resource utilization of a mesh is higher than that of a tree. Meshes based on Gossip protocol can find fresh peers in the single mesh with low management overhead, but not in global P2P networks. Due to the random selection algorithm, the quality of service cannot be guaranteed, such as the startup delay. Also, to decrease the impact of autonomy of peers on streaming services, very large buffer space, such as used in Coolstreaming, is necessary.

Zhang proposed a DHT based P2P resource pool, SOMO [22], [23] to manage global resources and optimize multiple ALM (Application Layer Multicast) sessions, especially computation applications. The main idea of such approaches is to structure all peers strictly [24], ignoring the features of specific applications. However the huge maintenance overhead makes these approaches far from scalable. Indeed, even if we have global knowledge of a P2P network, finding an optimal assignment of resources is still NP-hard. Based on a completely distributed heuristic, our proposed approach selects streaming paths and uses key links or peers as backup providers. Inter-overlay optimization is conducted in AnySee to complement traditional intra-overlay strategies.

III. ANYSEE DESIGN

To achieve good performance in P2P live streaming systems, AnySee faces the following challenges: (1) how to find paths with low delays, including source-to-end delay and startup delay, in a global P2P network; (2) how to maintain the service continuity and stability (decreasing the impact of interruption caused by peers leaving); (3) how to determine the frequency of optimization operations; and (4) how to reduce the control overhead caused by the algorithm. We introduce the design of AnySee in this section.

A. Overview

As illustrated in Fig. 3, the basic workflow of AnySee is as follows. First, an efficient mesh-based overlay is constructed. A location detector based algorithm is employed to match the overlay with the underlying physical topology [25]. Second, the *single overlay manager*, which is based on traditional intra-overlay optimization, such as Narada [13] and DONet, deals with the join/leave operations of peers. Third, the *inter-overlay optimization manager* explores appropriate paths, builds backup links, and cuts off paths with low QoS for each end peer. Fourth, *the key node manager* allocates the limited resources, and the *buffer manager* manages and schedules the transmission of media data.

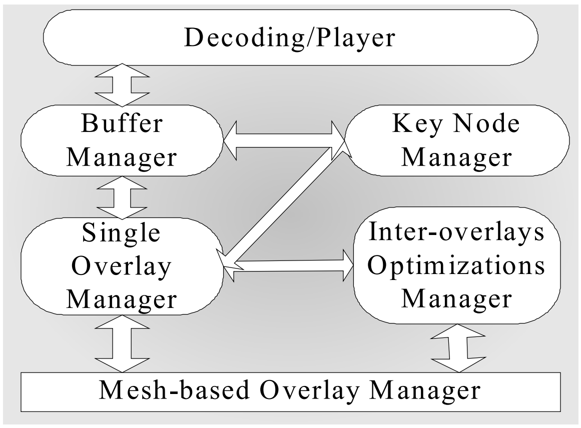


Figure 3. The system diagram of an AnySee node

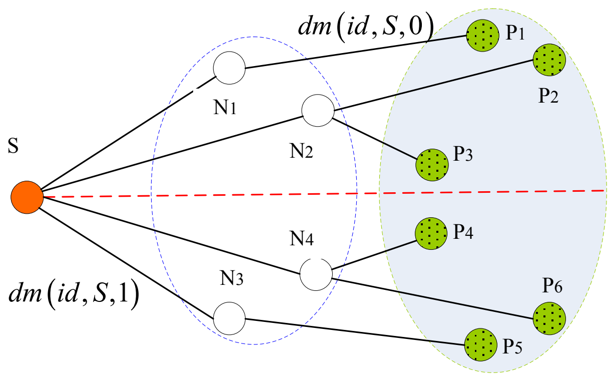


Figure 4. Roadmap of detector message initaited by S

B. Mesh-based Overlay Manager

In AnySee, peers join the mesh-based overlay first. Every peer, with a unique identifier, first connects the bootstrapping peers and selects one or several peers to construct logical links. Every peer maintains a group of logical neighbors. The key issue here is to let the mesh-based overlay match with the underlying physical topology [26] . The *mesh-based overlay manager*, a key component of AnySee, uses some strategies, such as an LTM (Location-aware Topology Matching) technique [25], to optimize the overlay, find the latest neighbors, and eliminate slow connections. There are two major operations: flooding-based detection with limited TTL, and updating logical connections.

In the first operation, each peer periodically floods a message, defined as *dm*(*id*, *S*, *TTL*), to its neighbors. The message *dm*(*id*, *S*, *TTL*) means that the peer initiates a message with ID value *id* in *TTL* hops. Since our purpose is to find the latest neighbors of peer *S*, we define TTL=2. To detect the distance of peers, the message body has six parts, including *messageID*, *TTL* value, *sourceIP* (the IP address of the source peer), *sourceTimestamp* (the timestamp[1] when the source forwards the message), *DirectIP* (the IP address of one neighbor within one hop) and *DirectTimestamp* (the timestamp when the neighbor within one hop gets the message). Figure 4 shows the roadmap of one message from *S*. Obviously, a message is broadcast to direct neighbors and 2-hop away neighbors.

In the second step, logical links are updated. With the help of the timestamps on peers, peer *P1* compares the distance between two paths, $S \rightarrow P1$ *and* $S \rightarrow N1 \rightarrow P1$. If the former length is larger, the link $N1 \rightarrow P1$ would be cut off and the direct path between *S* and *P1* would be established. All peers would do the same operations as those of peer *S*. After several operations, peers would connect with their nearest neighbors.

[1]The clocks of all peers are synchronized based on NTP. Current implementation of NTP version 4.1.1 in public domain can reach the synchronization accuracy down to 7.5 milliseconds [27].

C. Single Overlay Manager

The *single overlay manager* is responsible for peers leaving/joining operations. Before inter-overlay optimization, one peer joins one streaming overlay and receives media contents from multiple providers or single provider according to intra-overlay optimization schemes. In this design, a new attribute is introduced called *LastDelay*, which is the minimal of all source-to-end delays from the current node to the streaming source on different paths. With *LastDelay*, each path to the media source can be measured and evaluated. When a media block is delivered from the media source to the node, the *single overlay manager* records the timestamp and writes it into the media block's header. When a peer receives the media block with the initial timestamp, it computes the difference of the initial timestamp and the arriving timestamp. The minimal difference is the value of *LastDelay*. Peers can join or leave the topology according to *LastDelay*.

D. Inter-overlay Optimization Manager

Generally, each peer maintains one active streaming path set and one backup streaming path set. Initially all streaming paths are managed by the *single overlay manager*.

When the number of backup streaming paths is less than a threshold, the inter-overlay optimization algorithm is called to find appropriate streaming paths in the global P2P network with the help of the mesh-based overlay. When one active streaming path is cut off due to its poor QoS or peer's leaving, a new streaming path is selected from the backup set.

Basically, a peer *P* under source *S* with a streaming rate *rate*(*S*) maintains (1) an active streaming path set with threshold size $\delta_a(P,S)$, and (2) a backup streaming path set with threshold size $\delta_b(P,S)$. Each streaming path SP_i from *S* to *P* has two parameters: *delay* (*SP*$_i$, *S, P*) is the source-to-end delay from source S to peer P; $rate(SP_i, S, P)$ is the streaming rate in the last hop of the path. Clearly, we have:

$$\sum_{i=1}^{\delta_a(P,S)} rate(SP_i,S,P) \ge rate(S) \qquad (1)$$

$$\sum_{i=1}^{\delta_b(P,S)} rate(SP_i,S,P) = p\sum_{i=1}^{\delta_a(P,S)} rate(SP_i,S,P) \qquad (2)$$

Let $\mu_D(S)$ denote the threshold for the delay, which is related to the priority of the streams only.

We also design a probing message named *ProbM* as shown in Fig. 5. This message includes two major parts: (1) initial information about the message, including sequence number, *Seq.*, initial peer ID, *Peer_0*, message issuance time, *Timestamp0*, media source ID of the initial peer, *Source*, current *LastDelay*, and TTL; (2) an array with the size of TTL to record peer ID and the arriving timestamp of the message. Considering that 95% of peers in the Gnutella system could be reached within 7 hops by pure flooding, the maximal TTL is set to 7.

There are mainly two tasks for the *inter-overlay optimization manager*, including backup streaming path set management and active streaming path set management.

The major operation in backup streaming path set management is the probing procedure, called *reverse tracing* algorithm. This algorithm starts when the size of backup set is less than $\delta_b(P,S)$. *Peer_0* sends out a *ProbM* message to *j* of its neighbors with the recording array empty. Each receiver records the message arrival time and its ID into the accepted message body. The receiver will stop forwarding the message if (1) it finds that the delay from the initial peer *Peer_0* to this peer is greater than *LastDelay*; or (2) the receiver is the source of this streaming service. Otherwise, the message would be forwarded to *j* random neighbors.

After *reverse tracing*, the media source is able to analyze the arrived messages with ID Seq., and explore the best path from the source to the message issuance peer. Informed by the source, the peer is able to construct the best overlay path accordingly. Figure 6 shows an example of the reverse tracing algorithm based on the overlay shown in Fig. 1, when *j*=3 and *j*=2, respectively. In this figure, all delays are replaced with the cost of two peers. Peer D sends out a message and the possible routes of the message are illustrated. Some routes are cancelled due to a longer delay than *LastDelay*. Eventually, a good path S1→S2→D is successfully selected. Then *LastDelay* is updated. As a large portion of *ProM* messages are stopped during forwarding process, the overhead is acceptable.

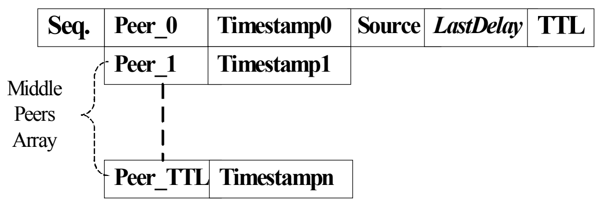
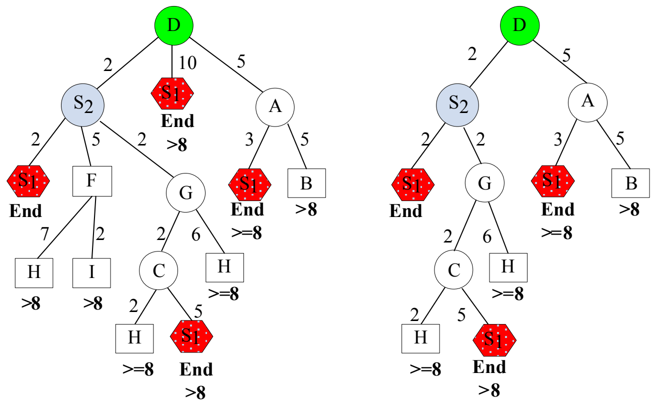


Figure 5. Structure of message *ProbM*



(a) $LastDelay = 8$
j=3

(b) $LastDelay = 8$
j=2

Figure 6. Examples of reverse tracing algorithm: (a) each peer forwards to three neighbors; (b) each peer forwards to two neighbors

A streaming path is treated as invalid if (1) the source-to-end delay is larger than a given threshold $\mu_D(S)$, or (2) the direct parent of the end peer on the path leaves. In this design, we only disconnect the overlay link between the end peer to its parent node because (1) the other connections on the path can be reserved to provide support for new incoming peers, and (2) our observations show that large delays often come from the last connection in the path, and (3) frequent disconnections incur a lot of unnecessary traffic.

The management of an active streaming path set has three operations, including maintaining the states of active streaming paths, cutting off invalid paths, and adding new active paths from a backup set, which are straightforward. When the total bit rates from active streaming paths are lower than *rate*(*S*), the manager will check whether a better path should be activated to replace the current one.

This manager has the following characteristics: (1) it employs a heuristic algorithm, and the system is optimized step by step; (2) probing procedures have originated from the normal peers, not the source peer, so that the control overhead is balanced to normal peers; (3) the number of forwarding neighbors, *j*, balance the tradeoff between the optimization effectiveness and the overhead; (4) the frequency of probing and optimization is dynamic. In AnySee, probing procedure is feedback-driven based on delay. Here how to set the initial value of the threshold, $\mu_D(S)$, is of importance. Logs from AnySee, to be described in Figures 19-21, show that when peers are watching highly popular movies, they are willing to tolerate a higher delay as much as 30 seconds. It is reasonable that different programs define different $\mu_D(S)$.

E. Key Node Manager

It is of great importance for peers to have an effective admission control policy when there are too many requests. Suppose each peer has *N* spare connections. According to the characteristics of requests, each request will fall into one of *M* queues with different priorities and popularities. When we assign the *N* spare connections to *M* queues, there are two interesting cases. First, some queues are assigned with more than one connection tunnel, which can be modeled as an M/M/m/K queuing system [28]. Second, some queues only receive one connection, which follows an M/M/1/K queuing model.

The admission control policy of a peer is designed to make the resources utilization optimal. The problem can be described as follows. Suppose there are M queues of requests. The arriving rate of queue j is λ_j, all arriving rates satisfy $\lambda_1 < ... < \lambda_j < ... < \lambda_M$. The service rate to assign one connection is μ and each connection processor can buffer k requests ($k \geq 1$). Assuming the probability that n requests follow the M/M/m/K queuing model is p_n, we have

$$p_n = \begin{cases} \dfrac{(m\rho)^n}{n!} p_0 & n = 0,1...m-1 \\ \dfrac{m^m \rho^n}{m!} p_0 & n = m, m+1...K \end{cases} \tag{3}$$

where $\rho = \dfrac{\lambda}{m\mu}$; we also have

$$p_0 = \begin{cases} \left[\left[\sum_{i=0}^{m-1} \dfrac{(m\rho)^i}{i!} + \dfrac{(m\rho)^m}{m!} \dfrac{1-\rho^{K-m+1}}{1-\rho} \right] \right]^{-1} & \rho \neq 1 \\ \left[\left[\sum_{i=0}^{m-1} \dfrac{(m)^i}{i!} + \dfrac{(m)^m}{m!} (K-m+1) \right] \right]^{-1} & \rho = 1 \end{cases} \tag{4}$$

Thus, the average utilization of N spare connections of one peer can be given by:

$$\bar{\rho} = \rho(1-\rho k) = \rho\left(1 - \frac{m^m \rho^K p_0}{m!}\right) \tag{5}$$

One connection processor can buffer k requests, then $K = mk$. When the probability that n requests are following the M/M/1/K queuing model is $p_n^{'}$, we have

$$p_n^{'} = \begin{cases} \dfrac{(1-\rho)\rho^n}{1-\rho^{K+1}} & \rho \neq 1 \\ \dfrac{1}{K+1} & \rho = 1 \end{cases} \quad 0 \leq n \leq K \tag{6}$$

and $\rho = \dfrac{\lambda}{\mu}$. Then the average utilization of N spare connections of one peer can be given by

$$\tilde{\rho} = 1 - p_0^{'} = \rho\left(\frac{1-\rho^K}{1-\rho^{K+1}}\right) \tag{7}$$

and $p_K^{'}$ is the failure probability of requests. Then, the target can be expressed:

$$Max\left(\rho(N_1, N_2, ...N_M)\right) = Max\left(\sum_{1 \leq i,j \leq M}^{i \neq k} \left(\bar{\rho}_i + \tilde{\rho}_j \right) \right) \tag{8}$$

$$Subject\ to \quad \sum_{i=1}^{M} N_i = N \quad 1 \leq N_i < N$$

The above optimization problem (Eq. (8)) can be divided into two parts. First, we enumerate all (M, 1)-partitions (M queues and each should be allocated at least 1 connection) of N spare connections such that the best allocation can be found to maximize $\rho(N_1, N_2, ...N_M)$ in Eq. (8). Second, for all H partitions of N connections, we can compute all H results of average resources utilization and select the best partition, based on which of the resources utilization is maximal. In the first phase, we can get H, the number of partitions of N by

$$H = \binom{N-1}{M-1} = \frac{(N-1)!}{(M-1)!(N-M)!}, N \geq M \tag{9}$$

From Equation (9), the first algorithm complexity is O(N). The second algorithm is to select the maximal one from H results. Its complexity is $O\left(C_{M-1}^{N-1}\right) = O(N)$. Consequently, this optimization problem has complexity of O(N). Considering one normal peer with 10Mbps bandwidth and average streaming rate 300Kbps, N should be set less than 33.

F. Buffer Manager

This manager is responsible for receiving valid media data from multiple providers in the active streaming path set and continuously keeping the media playback. AnySee employs a similar heuristic as used in the Coolstreaming system [12] to fetch expected media segments in a dynamic and heterogeneous network to meet two constraints: the playback deadline for each segment and the heterogeneous streaming bandwidth from partners. As Coolstraming does not employ any inter-overlay optimization, peers often fail to find the closest neighbors to supply services. To keep the media playback continuous, a big buffer must be used. Due to the effectiveness of the inter-overlay optimization scheme adopted in AnySee, a small buffer space is enough, and indeed a small buffer often means a shorter startup delay.

IV. SIMULATION

Before introducing our implementation experiences and the observation about the real AnySee system, we evaluate AnySee with comprehensive simulations and contrast its performance with a recent live streaming system, Coolstraming [12].

A. Simulation Methodology

We consider two types of topologies, physical topology and logical P2P topology. The physical topology represents a real topology with Internet characteristics. The logical topology represents the overlay P2P topology built on top of the physical topology. All P2P nodes are in a subset of nodes in the physical topology. The communication cost between two logical neighbors is calculated based on the shortest physical path between this pair of nodes.

We develop a crawler based on Gnutella protocol [1] and the source codes are rewritten from Limewire open source client [2]. The crawler's main function is to probe the connections of Gnutella peers. When peers are receiving crawler ping messages, they reply with corresponding pong messages. With the help of forty-five independent threads, our crawler discovers over fifty thousands peers and their connections in one week. In this simulation we use three data sets, obtained from different time slots. Each trace includes around 2,000 peering nodes.

For the physical topology, we use BRITE [3] generating three topologies, each with 5,000 nodes. The average number of neighbors of each node ranges from 4 to 10.

The major parameters in our simulations are listed in Table 1. In each run, peers randomly join one of *S* streaming overlays (S=1, 4, 8, 12). Each peer randomly has *C* connections ranging from 4 (1 Mbps bandwidth) to 40 (10 Mbps bandwidth) and maintains at least *M* neighbors. The size of each overlay is *N* (N<500). Each stream is 1800-seconds long, and the streaming rate is *r*, normally 300 Kbps. Based on the delay values from the trace, we set the bandwidths for peers. For simplicity, the threshold $\mu_D(S)$ is set to 25 seconds, which is estimated from logs of AnySee implementation. The adjustment factor p is set to 1, which means we provide one backup streaming path for each active streaming path.

To better evaluate the performance of AnySee, we use the metrics as follows. (1) *Resource utilization* is defined as the ratio between the used connections to all connections; (2) *Continuity* index, representing the playback continuity, is defined as the number of segments that arrive before playback deadlines over the total number of the segments.

TABLE I. SIMULATION PARAMETERS

Abbreviate	Comment
S	Number of streaming overlays
M	Number of neighbors
N	Size of one overlay
r	Streaming playback rate
C	Number of total bandwidth connections

B. Results

The first set of simulations is conducted in a stable environment, in which peers do not leave after joining the overlays. For each simulation setup, we take 100 runs and report the average.

We first evaluate the QoS of the AnySee service in a stable environment. Figure 7 plots the continuity index against streaming rate, where we contrast AnySee and Coolstreaming. When the streaming rate is increased, the continuity of AnySee is relatively good while the continuity of Coolstreaming is degraded. There are two reasons. First, AnySee can find more near neighbors from all peering nodes to request services, while Coolstraming is only able to find suppliers from the same overlay. Second, the necessary buffer size of AnySee is only 40 seconds, while Coolstreaming needs a 120-second buffer.

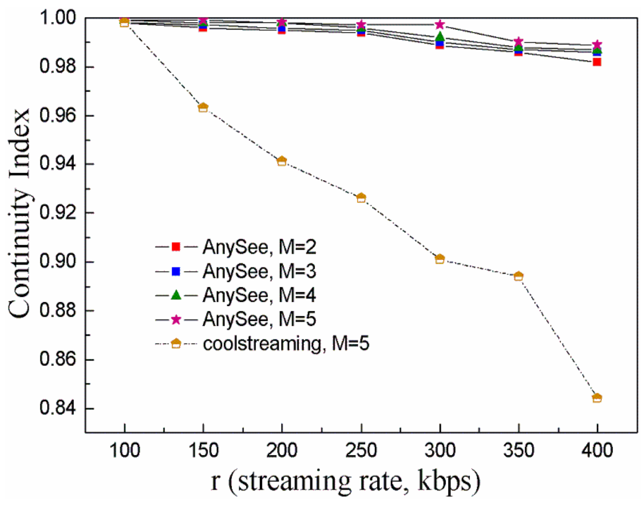


Figure 7. Continuity index V.S. streaming rates when N=400, S=12 and initial buffer size is 40 seconds

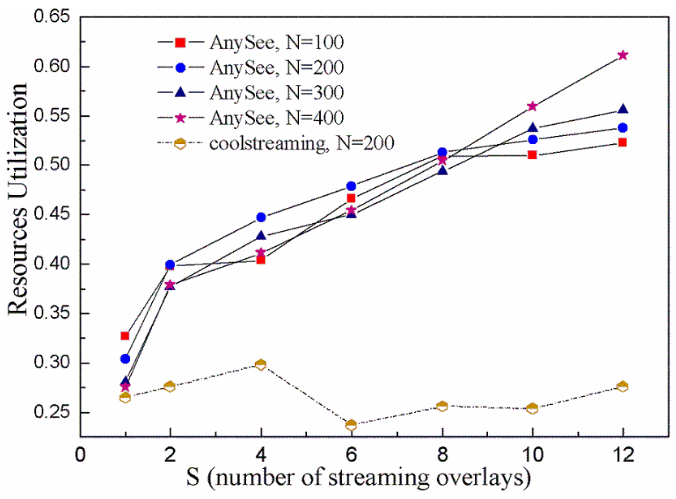


Figure 8. Resources utilization: overlay size V.S. the number of streaming overlays when M=12, r=300 Kbps

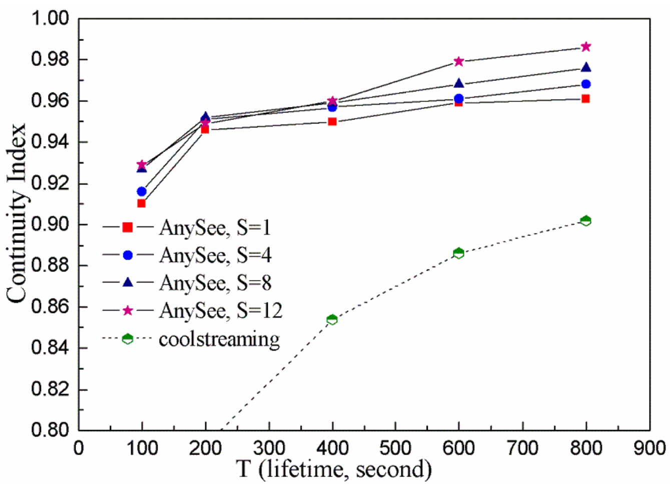


Figure 9. Continuity index under dynamic environments when M=5, N=400, r=300 Kbps and initial buffer size is 40 seconds

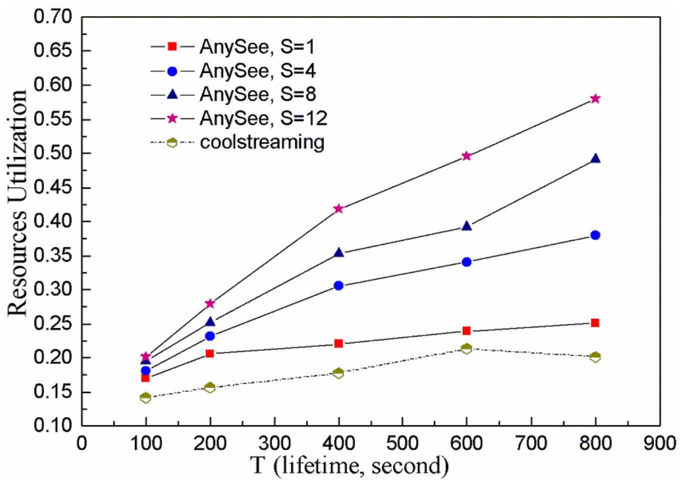


Figure 10. Resource utilization under dynamic environments when M=5, N=400, and r=300 Kbps

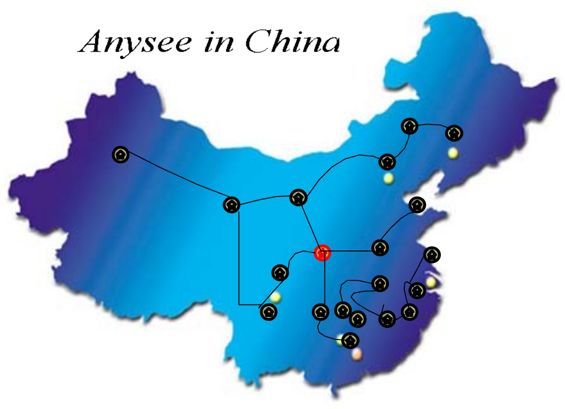


Figure 11. Services map of AnySee in CERNET of China (red center point: HUST, Wuhan)

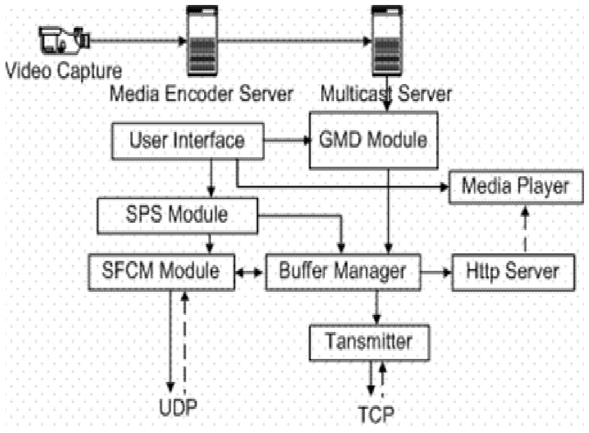


Figure 12. System modules of AnySee

Figure 8 contrasts resource utilization of AnySee and Coolstreaming. Seen on the left part of Figure 9, a larger number of streaming overlays has a greater impact on the performance of AnySee, but no obvious influence on Coolstreaming. This is due to the fact that Coolstreaming does not let peers to select better relay paths using other peers in different overlays.

We then conduct simulations when peers are leaving and joining freely. We define the lifetime of each peer in the overlay, from 100 seconds to 500 seconds. Peer average lifetime is exponentially distributed with an average of T seconds. We can see from Figures 9 and 10 that longer lifetime leads to better service quality and higher resource utilization. However, when the average lifetime of peers is short, the continuity of Coolstreaming is relatively poor. As our proposed AnySee has a backup path management design and the reverse tracing component keeps finding better paths dynamically, AnySee always outperforms Coolstreaming.

V. IMPLEMENTATION OF ANYSEE

We have implemented the public free system, AnySee, and released two versions (v.1.0 and v.1.1) to provide a scalable live-streaming service platform based on inter-overlay optimization in CERNET of China. From June 2004 to February 2005, there were over 60,000 connections to the platform and above 40 universities and 20 cities in China were in the service map as shown in Fig. 11. The system is implemented with Java and is platform-independent.

A. *Architecture Overview*

AnySee system is comprised of four components. They are (1) a rendezvous point (RP), (2) a media source, (3) a monitor, and (4) end systems. Each end system contains an *IP to Network Coordinates Database* (INCD), which is pre-built and integrated into the end system software.

Figure 12 shows the modules of end systems in AnySee. Every end system (including the Broadcaster) is composed of several function modules as follows: (1) *getting media data* (GMD) module is for Broadcaster; (2) *sending peer selection* (SPS) module is deployed on all peers except the Broadcaster; (3) *session for controlling message* (SFCM) module is responsible for exchanging control messages between current peer and its supplier, and monitoring actions of child peers; (4) *buffer manager* (BM) module gets media packets from the upper-layer, sends them to the HTTP server module, and deletes packets with outdated timestamps in the buffer; (5) *data transmitter* (DT) module fetches media packets from the buffer, and transmits packets to underlying peers under flow control policy; (6) *HTTP server* (HS) module creates a virtual HTTP service at a local machine. After retrieving media data packets from the buffer, HS module sends them to media players such as Windows Media Player, under the HTTP protocol.

B. *Implementation Experiences*

We discuss two interesting issues in AnySee implementation, GID based service scheme and locality-aware buffer management scheme.

1) *GID based service scheme*

Due to the characteristics of streaming applications, it is desirable to let every peer to get media services from suppliers with low latency and high bandwidth. Many approaches have been proposed, such as GNP. However, most of them are too complex to be feasible. In AnySee, all peers are in the same CERNET and the physical network map is well known. It is efficient that the distance is computed with the help of the paired IP addresses. Thus, AnySee requires each peer maintain an INCD, from which each peer can have a position, named GID in the global network. The GID value of an end host is a 128-bits integer encoded by the 4-layer geometrical information corresponding to ISPs, cities, campuses, and buildings, respectively. Such information is also used by AnySee to estimate the physical locations of peers.

2) *Locality-aware buffer management scheme*

As the behavior of peers in upper layers have a larger impact on QoS than that of peers in the lower layers, AnySee employs a layer-aware buffer management scheme. Each peer computes its appropriate buffer space size according to the layer number. In AnySee, the buffer size of peer *A* at the *m*-th layer is given by:

$$T_A = f(m) = \varepsilon \times t_A + t_A' \qquad (10)$$

where t_A denotes the total link delay, t_A' is the total transporting delay, and ε is the average disconnection times of one connection.

Suppose the probability of a link or node failure is P_b, and a peer needs t_b time to explore a new parent, the border delay is $t_l = P_b \times t_b$ and the link delay t_A is the accumulation of all border delays. Suppose the transporting delay per hop is μ and the total hops between the source peer and peer *A* is *m*, the total transporting delay of peer *A* is $t_A' = \mu \times m$. If the path from source peer to peer *A* is $l_{S\to A} = \{l_{S\to a_1}, l_{a_1\to a_2}, \ldots, l_{a_{m-1}\to A}\}$, the path has the following properties: (1) the source peer would persist all the time and the path $l_{S\to a_1}$ would not break down; (2) peer *A* would also stay in the network and the path $l_{a_{m-1}\to A}$ would exist; (3) the influence that multiple borders break down simultaneously is the accumulation of influence that multiple borders break down one by one; (4) if one peer a_i leaves, one new peer a_i' would join the tree and replace the position of a_i. Then the total link delay can be computed as

$$t_A = \sum_{i=1}^{i=m-1} t_{l_{a_i\to a_{i+1}}},$$

and

$$t_{l_{a_i\to a_{i+1}}} = (1-P_b)^{i-1} \times t_l$$

then, $t_A = t_l + (1-P_b)\times t_l + \ldots + (1-P_b)^{m-2} \times t_l$,
after computation,

$$t_A = t_l \times \frac{1-(1-P_b)^{m-1}}{P_b} = t_b \times \left(1-(1-P_b)^{m-1}\right) \qquad (11)$$

Then, we have

$$T_A = \varepsilon \times t_b \times \left(1-(1-P_b)^{m-1}\right) + \mu \times m \qquad (12)$$

Given the estimation of the above parameters in Eq. (12), the maximum buffer size of peer *A* at the *m*-th layer is computed which only relates to layer *m*.

C. *Performance of AnySee*

Among all log data collected, we select records from 13/08/2004 to 29/08/2004. Over 7200 users from over 40 universities in 14 cities of China received services with AnySee. We analyzed the performance of the multiple multicast trees every ten minutes.

Figure 13 plots the average height of AnySee trees against tree size. Although the height increases when more peers join each service tree, the height is always less than 7 even with a thousand peers included in one tree. Such a property helps shorten the source-to-end delay as shown in Fig. 14. We can see the source-to-end delay is always less than 200 ms. From the logs of AnySee, the startup delay of most peers is less than 20s. We have implemented a simple prototype, which can get media services from a Coolstreaming network, and we observe the startup delays for 50 times. Mostly, the startup delay of Coolstreaming is around 60 seconds. Based on the results shown in Fig. 14, we set μ to 20ms and define $t_b = 2$, $P_b = 0.4$, $\varepsilon = 2$.

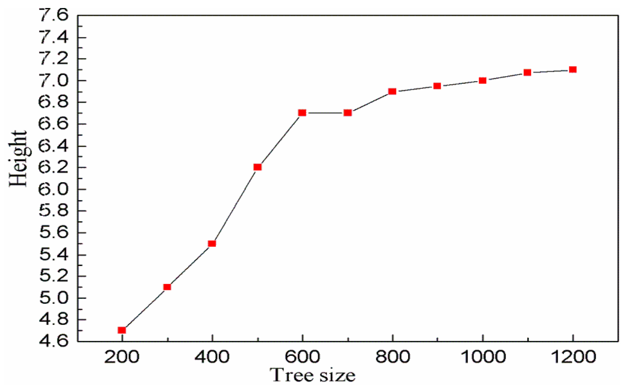


Figure 13. Height V.S. tree size

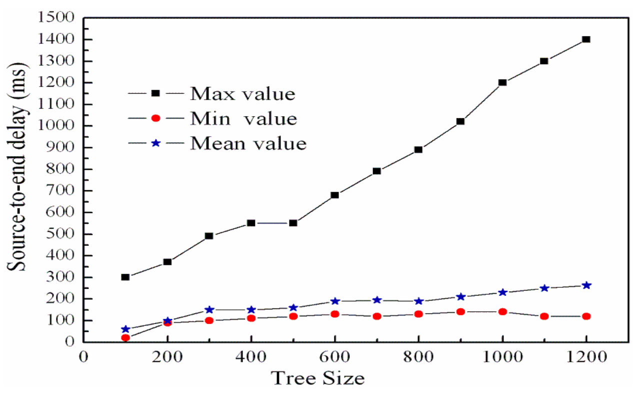


Figure 14. Source-to-end delay V.S. tree size

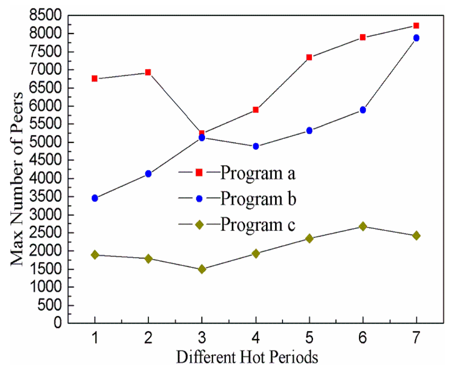


Figure 15. Maximum number of peers in different hot periods

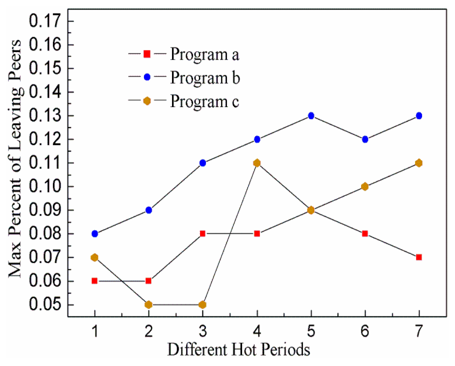


Figure 16. Maximum percentage of leaving peers in different hot periods

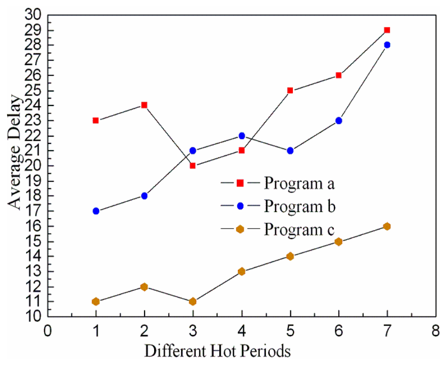


Figure 17. Average delay in different hot periods

D. *Users Behavior and System Optimization*

It is important to know the user behavior, which can help us optimize the system. We select three log sets to analyze the total number of peers, the average delay, and the leaving peer percentage in different "hot" periods. We select 7 different hot periods for three programs, Program-a, Program-b, and Program-c.

Figures 15, 16, and 17 show the maximum number of peers, maximum percentage of leaving peers, and average delay of three programs, including Program-a, Program-b, and Program-c for one hour. The results show that the overlays with popular movies attract more users to join, but cause larger average delays. From the figures, we have the following interesting observations. First, larger delay is not always the major reason that causes people to leave the overlay. For example, the leaving percentage of Program-a is not the largest while its average delay is the longest. Peers have more patience than that imaged by previous researchers if the program is very popular. Second, delays from 20 to 30 seconds will not be the killer for the live streaming services. Most people will still stay in the overlay even if there is a 30 second delay from the source peer.

Based on the above observations, AnySee does not determine the optimization frequency only according to the average delay, but also the percentage of leaving peers. That too many users are leaving the overlay is the signal that the overlay is under heavy burden and needs to be optimized. AnySee defines the parameter "optimization index", *ADL*, which is given by

$$ADL = 100 \times \frac{leaving\ percentage}{average\ delay}$$

TABLE II. *ADL*S IN HOT PERIODS OF DIFFERENT PROGRAMS

Num.	Program-a	Program-b	Program-c
1	0.5217	0.4706	0.6363
2	0.4583	0.5000	0.4167
3	0.7000	0.5238	0.4545
4	0.7143	0.5455	0.8462
5	0.5600	0.6190	0.6429
6	0.6154	0.5217	0.6667
7	0.5862	0.4642	0.6875
Average	**0.5937**	**0.5213**	**0.6215**

After computation, three *ADL*s from different periods are shown in Table 2. From Table 2, the average *ADL*s for the above programs are 0.5937, 0.5213, and 0.6215, respectively. AnySee provides a threshold on ADL to determine whether an overlay optimization is necessary.

VI. Conclusion and Future Work

Efficient and scalable live-streaming overlay construction has become a hot topic recently. In order to improve the metrics, such as startup delay, source-to-end delay, and playback continuity, most previous studies focused on intra-overlay optimization. Such approaches have drawbacks including low resource utilization, high startup and source-to-end delay, and inefficient resource assignment in global P2P networks.

In this paper, we propose an inter-overlay optimization based live streaming scheme. Instead of selecting better paths in the same overlay, AnySee peers are able to construct efficient paths using peers in different overlays. We evaluate the performance of AnySee by comprehensive simulations. Our experimental results show that AnySee outperforms existing intra-overlay live streaming schemes, such as Coolstreaming.

The practical AnySee system has been released for several months and its client code is free to be downloaded in CERNET of China. To date, over 60,000 users benefit from AnySee to enjoy two international academic conferences, namely GCC'04 (Grid and Cooperative Computing) and NPC'04 (Network and Parallel Computing), and other massive entertainment programs. Logs from AnySee show that users have great patience for live streaming services with large delay if they have enough interest in the programs. We hope the system can serve more people and attain better quality in the future.

We are currently building peer-to-peer video-on-demand services for large-scale users based on inter-overlay optimization schemes. We are going to observe more user behaviors to further improve the system performance.

Acknowledgment

This work was partially supported by China National Natural Science Foundation (NSFC) under grant No.60125208, 60433040, 60573129, 60573140, Specialized Research Fund for the Ph.D Program from Ministry of Education under grant No. 20050487040, and Hong Kong RGC Grants HKUST6264/04E, DAG 05/06.EG44 and AoE/E-01/99.

References

[1] The Gnutella protocol specification 0.6, http://rfc-gnutella.sourceforge.net.
[2] Limewire, http://www.limewire.com/.
[3] BRITE, http://www.cs.bu.edu/brite/.
[4] J. Liu, B. Li, and Y.-Q Zhang, "Adaptive Video Multicast Over the Internet", *IEEE Multimedia*, 2003.
[5] B. Alfonsi, "I Want My IPTV: Internet Protocol Television Predicted a Winner", *IEEE Distributed Systems Online*, 2005.
[6] R. Perlman, "Models for IP Multicast", in Proceedings of IEEE International Conference on Networks, 2004.
[7] A. Ganjam and H. Zhang, "Internet Multicast Video Delivery", *IEEE Proceeding*, 2005.
[8] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-Time Applications", in Proceedings of IEEE INFOCOM, 2003.
[9] A. Myers, T.S.E. Ng, and H. Zhang, "Rethinking the Service Model: Scaling Ethernet to a Million Nodes", in Proceedings of ACM SIGCOMM HotNets*,* 2004.
[10] Y. Liu, X. Liu, L. Xiao, L. Ni, and X. Zhang, "Location-Aware Topology Matching in P2P Systems", in Proceedings of IEEE INFOCOM, 2004
[11] T.S.E. Ng and H. Zhang, "A Network Positioning System for the Internet", in Proceedings of USENIX, 2004.
[12] X. Zhang, J. Liu, B. Li, and T. P. Yum, "DONET: A Data-Driven Overlay Network for Efficient Live Media Streaming", in Proceedings of IEEE INFOCOM, 2005.
[13] Y. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," in Proceedings of ACM SIGMETRICS, 2000.
[14] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth Content Distribution in Cooperative Environments", in Proceedings of ACM SOSP, 2003.
[15] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh", in Proceedings of ACM SOSP, 2003.
[16] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking", in Proceedings of ACM NOSSDAV, 2002.
[17] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "PROMISE: Peer-To-Peer Media Streaming Using Collectcast", in Proceedings of ACM Multimedia, 2003.
[18] D. Tran, K. Hua, and S. Sheu, "Zigzag: An Efficient Peer-To-Peer Scheme for Media Streaming", in Proceedings of IEEE INFOCOM, 2003.
[19] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast", in Proceedings of ACM SIGCOMM, 2002.
[20] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: P2P Patching Scheme for VoD Service", in Proceedings of WWW, 2003.
[21] X. Jiang, Y. Dong, and X. D, B. Bhargava, "GNUSTREAM: A P2P Media Streaming System Prototype", in Proceedings of IEEE ICME, 2003.
[22] Z. Zhang, Y. Chen, S. Lin, B. Lu, S. Shi, X. Xie, and C. Yuan, "P2P Resource Pool and Its Application to Optimize Wide-Area Application Level Multicasting", in Proceedings of International Conference on Parallel Processing Workshops, 2004.
[23] Z. Zhang, S. Shi, and J. Zhu, "SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT", in Proceedings of IEEE IPTPS, 2003.
[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network", in Proceedings of ACM SIGCOMM, 2001.
[25] Y. Liu, L. Xiao, X. Liu, L.M. Ni, and X. Zhang, "Location Awareness in Unstructured Peer-To-Peer Systems", *IEEE Transactions on Parallel and Distributed Systems*, 2005.
[26] M. Ripeanu and I. Foster, "Mapping Gnutella Network", *IEEE Internet Computing*, 2002.
[27] NTP: The Network Time Protocol, http://www.ntp.org/.
[28] L. Kleinrock, Queueing Systems, John Wiley, 1974.
[29] Y. Liu, A-H. Esfahanian, L. Xiao, and L. M. Ni, "Approaching Optimal Peer-to-Peer Overlays", in Proceedings of IEEE MASCOTS, 2005.