

Scalable Live Streaming Service Based on Inter-Overlay Optimization

Xiaofei Liao, *Member, IEEE*, Hai Jin, *Senior Member, IEEE*,
 Yunhao Liu, *Senior Member, IEEE*, and Lionel M. Ni, *Fellow, IEEE*

Abstract—In order to provide scalable live-streaming services, we propose an Inter-Overlay Optimization scheme, IOO. Instead of selecting better paths in the same overlay, IOO constructs efficient paths using peers in different overlays, so as to (i) improve global resource utilization of P2P streaming networks; (ii) assign resources based on their locality and delay; (iii) guarantee streaming service quality by using the nearest peers, even when such peers might belong to different overlays; and (iv) balance the load among the group (streaming overlay) members. We compare the performance of IOO with existing approaches through trace driven simulations. Results show that IOO outperforms previous schemes in terms of resource utilization and the QoS of streaming services. IOO scheme has been implemented in an Internet based live streaming system, called AnySee. AnySee 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 videos.

Index Terms—Peer-to-peer, live streaming, inter-overlay optimization, startup delay, P2P topology discovery.

1 INTRODUCTION

With the improvement of network bandwidth, multimedia services based on streaming live media have gained much attention. 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 [8][9]. Due to the practical issues of routers, IP multicast has not been widely deployed. Therefore, researchers have expended a lot of efforts building efficient overlay multicast schemes based on P2P models, in which spectators behave as routers for other users. Scalable live-streaming overlay constructions [5] become a hot topic. Different from traditional distributed systems, streaming overlays focus on the following four metrics: startup delay, the delay to display the first image for users, including the logical media data transmission time and decoding preparing time; source-to-end delay, the delay to receive the first media data block directly or indirectly from the source; playback continuity, the number of segments that arrive before playback deadlines over the total number of the segments; and resource utilization, the ratio between the used bandwidth to the total bandwidth. These metrics have a direct bearing on the interactive usability of a live streaming system. Large delays would exhaust user patience and unplanned interruptions will spoil the entertainment value.

In order to improve the above metrics, previous studies [6] focused on intra-overlay optimizations, in which each

node joins at most one overlay. With the help of locality-aware strategies [7] and optimization schemes such as DO-Net in Coolstreaming [8], Narada in ESM [9], the QoS of live streaming P2Ps have significantly improved. Nevertheless, they still suffer from long delay and unplanned interruptions, especially when a large number of peers join the network simultaneously.

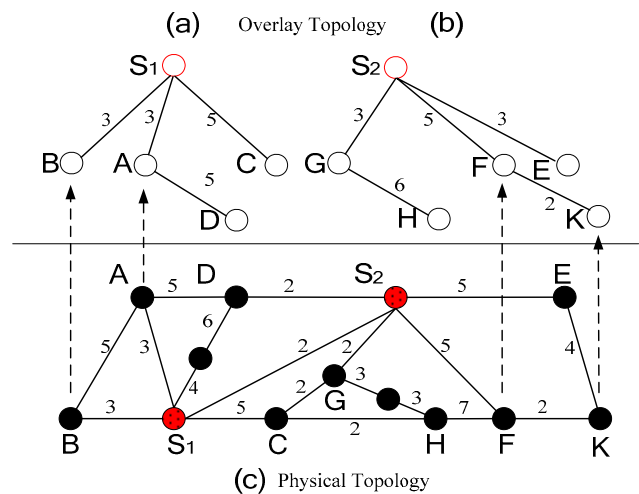


Fig. 1. Intra-overlay optimization: (a) optimal multicast tree rooted at S_1 ; (b) optimal multicast tree rooted at S_2 ; (c) physical topology.

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 are established as shown in Fig. 1(a) and Fig. 1(b), respectively. There are two obvious drawbacks. First, such overlay construction is not globally optimal. Considering

- Xiaofei Liao and Hai Jin are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. E-mail: xfliao@hust.edu.cn, and hjin@hust.edu.cn.
- Yunhao Liu and Lionel M. Ni are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong. E-mail: liu@cse.ust.hk, and ni@cse.ust.hk.

Manuscript received 11, August 2005; revised 19, April 2007; accepted 15, May 2007. Recommended for acceptance by Denis Trystram.
 For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0225-0806.

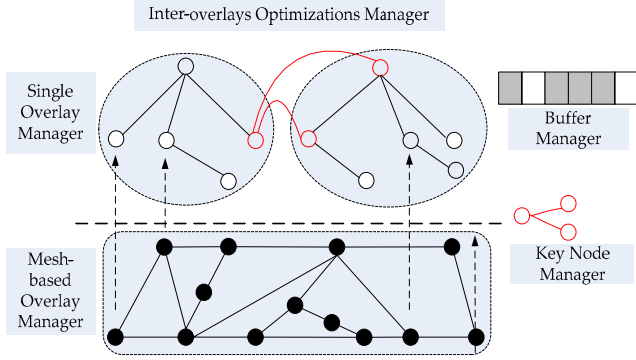


Fig. 3. Concept structure of IOO.

streaming tasks traditionally performed by media servers. 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 algorithm for neighbors' selection, the startup delay cannot be guaranteed. Also, to decrease the impact from autonomy of peers, a large buffer space, as used in Coolstreaming [8], is often necessary.

2.3 Hybrid overlays

Zhang et al proposed a DHT based P2P resource pool, SOMO [20], to manage global resources and optimize multiple ALM (Application Layer Multicast) sessions. The main idea is to structure all peers strictly [21], ignoring the features of specific applications. The huge maintenance overhead, however, makes these approaches far from scalable. Indeed, even if we have global knowledge of a P2P network, finding an optimal assignment of resources is NP-hard.

Based on a completely distributed heuristic, our proposed IOO scheme selects streaming paths and uses backup links or peers potential providers. Inter-overlay optimization is employed to complement traditional intra-overlay strategies.

3 DESIGN OF SCHEME

To achieve good performance in P2P live streaming systems, IOO scheme faces the following challenges: (i) to find paths with low delays, including source-to-end delay and startup delay, in a large-scale P2P network; (ii) to maintain the service continuity and stability, decreasing the impact of flash crowd and churn caused by peers leaving and joining, and network fluctuations; (iii) to determine the frequency of optimization operations; and (iv) to reduce the control overhead.

3.1 Overview

As illustrated in Fig. 3, the basic workflow of IOO scheme 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 [22]. Second, the single overlay manager, which is based on traditional intra-overlay optimization, such as Narada [9],

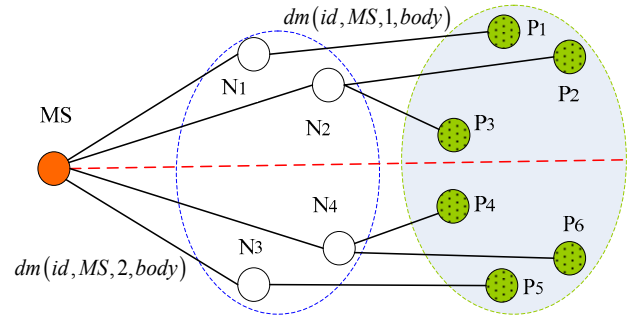


Fig. 4. Roadmap of detector message initiated by peer MS .

DONet or nearcast [26], 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.

3.2 Mesh-based Overlay Manager

In IOO, all peers belonging to different streaming overlays will join one substrate, mesh-based overlay first, to construct an under layer. In this layer, all peers can be reached each other directly or indirectly. The function of the mesh-based overlay manager is to provide information about the connections for the inter-overlay optimizations. 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 [23]. The mesh-based overlay manager, a key component of IOO, uses a scheme called as LTM (Location-aware Topology Matching) technique [22], to optimize the overlay, find the closest 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 $dm(id, MS, TTL, body)$, to its neighbors. The message $dm(id, MS, TTL, body)$ means that the peer MS initiates a message with id value to peers at most TTL hops away. Since our purpose is to find the closest neighbors of peer MS , we define $TTL=2$. To detect the distance of peers, the message body has four parts, including sourceIP (the IP address of the source peer), sourceTimestamp (the timestamp¹ 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 MS . Obviously, a message is broadcast to direct neighbors and 2-hop away neighbors.

¹The 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 [24].

In the second step, logical links are updated. With the help of the timestamps on peers, peer $P1$ compares the distance between two paths, $MS \rightarrow P1$, $MS \rightarrow N1$, and $N1 \rightarrow P1$. If the former length is smaller, the link $N1 \rightarrow P1$ will be cut off and the direct path between MS and $P1$ is established. All peers do the same operations as those of peer MS . After several operations, peers would connect with their close neighbors [22].

3.3 Single Overlay Manager

The single overlay manager is responsible for peers leaving and joining operations based on the under layer, mesh-based overlay. 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. Here we apply one scheme, named "locality-aware" tree based overlay construction scheme [26] to build optimized single overlay.

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.

Specifically, when a peer P is subscribing the media data from media source MS , MS will return the streaming rate $rate(MS)$ of the media program to peer P . Then P maintains (i) an active streaming path set with threshold size $\delta_a(P, MS)$, and (ii) a backup streaming path set with threshold size $\delta_b(P, MS)$. For the i -th streaming path SP_i from MS to P , it has two parameters: $delay(SP_i, MS, P)$ is the source-to-end delay from source MS to peer P on the path SP_i . When a media block is delivered from the media source to P on the path SP_i , the single overlay manager records the timestamp and stores it into the media block's header. Upon receiving the media block with the initial timestamp, P computes the difference of the initial timestamp and the arriving timestamp. The little difference is the value of $delay(SP_i, MS, P)$, and $rate(SP_i, MS, P)$ is the streaming rate in the last hop of the path. Clearly, we have:

$$\begin{aligned} \delta_a(P, MS) \sum_{i=1}^{P, MS} rate(SP_i, MS, P) &\geq rate(MS) \\ \delta_b(P, MS) \sum_{i=1}^{P, MS} rate(SP_i, MS, P) &= q \sum_{i=1}^{\delta_a(P, MS)} rate(SP_i, MS, P) \end{aligned} \quad (1)$$

In Eq.(1), the parameter q means that how many times the total downloading bandwidth from backup streaming path set is bigger than that from active streaming path set. Let $D(MS)$ denote the threshold for the delay, which is related to the requested quality of the streaming service only. The higher the requested quality of streaming service, the less the threshold of delay will be. 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. We have,

$$LastDelay = \text{Min}(delay(SP_i, MS, P)) \quad (2)$$

$$1 \leq i \leq \delta_a$$

With $LastDelay$, each path to the media source can be measured. Peers can evaluate QoS of streaming paths and adjust the directed neighbors according to $LastDelay$. The adjustment of streaming path set can be done with the

management of direct connected neighbors in the same overlay. Using Gnutella protocol, peers find new neighbor candidates. Comparing pair delays, the fastest neighbors are selected to build direct connections from the candidates. Based on the streaming path sets of all directly connected neighbors, new streaming path set can be created. Because each direct connected neighbor has maintained $x = \delta_a(P, MS) + \delta_b(P, MS)$ streaming paths and the delays to all y neighbors can be calculated, the new streaming paths with smallest source-to-end delays can be selected after the following calculations:

$$DS_y(x) = \begin{bmatrix} D_{1,1}, D_{1,2}, \dots, D_{1,x} \\ D_{2,1}, D_{2,2}, \dots, D_{2,x} \\ \vdots \\ D_{y,1}, D_{y,2}, \dots, D_{y,x} \end{bmatrix}, DS_y = \begin{bmatrix} D_1, D_1, \dots, D_1 \\ D_2, D_2, \dots, D_2 \\ \vdots \\ D_y, D_y, \dots, D_y \end{bmatrix} \quad (3)$$

$$DS(y \times x) = DS_y(x) + DS(y)$$

In Eq.(3), the parameter $D_{i,j}$ denotes the source-to-end delay on the streaming path j of neighbor i , D_i denotes the direct delay to the neighbor i , $DS_y(x)$ is the delays matrix of all neighbors' streaming paths. $DS(y \times x)$ is the delays matrix of all potential streaming paths. From $y \times x$ streaming paths, we can choose $\delta_a(P, MS)$ active streaming paths and $\delta_b(P, MS)$ backup streaming paths.

3.4 Inter-overlay Optimization Manager

When the number of backup streaming paths is less than a threshold, the IOO is conducted 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, a new streaming path is selected from the backup set.

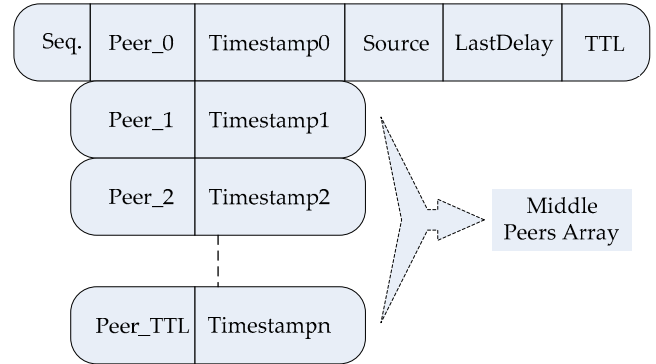


Fig. 5. Structure of the message *ProbM*.

We also design a probing message named *ProbM* as shown in Fig. 5. This message includes two major parts: (i) initial information, including *Seq.*: sequence number, *Peer_0*: initial peer ID, *Timestamp0*: message issuance time, *Source*: media source ID of the initial peer, *LastDelay*, current value of *LastDelay*, and *TTL*: current value of *TTL*; (ii) 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 [23], the maximal *TTL* is set to 7.

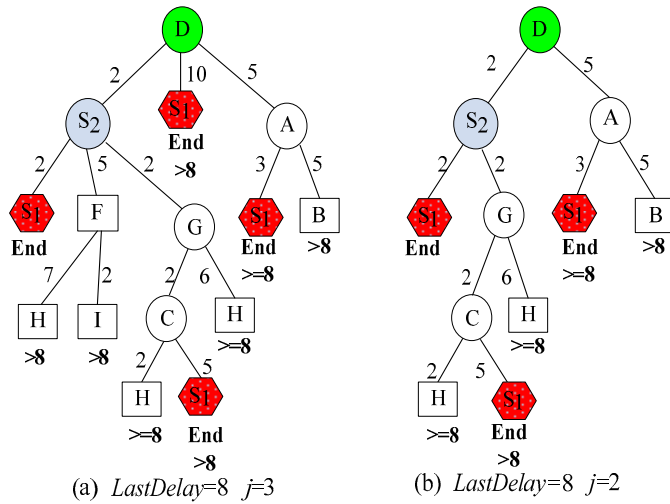


Fig. 6. An example of reverse tracing algorithm: (a) each peer forwards to three neighbors; (b) each peer forwards to two neighbors.

The inter-overlay optimization manager mainly has two tasks, backup streaming path set management and active streaming path set management.

- Managements of the backup streaming path set

The major operation in this component is a probing procedure, called reverse tracing algorithm. This algorithm starts when the size of the backup set is less than $\delta_b(P, MS)$. Peer_0 sends out a *Probm* message to j of its random neighbors with the recording array empty. Each receiver records the message arrival time and its ID into the message body of the received message, as shown in Fig. 5. The receiver will stop forwarding the message if (i) it finds that the delay from the initial peer Peer_0 to this peer is greater than $LastDelay$; or (ii) the receiver is the source of this streaming service. Otherwise, the message would be forwarded to j of its random neighbors.

With the help of reverse tracing algorithm, the media source is able to analyze the arrived messages with *ID (Seq.)* periodically, and explore the best path from the source to the message issuance peer. Informed by the source, the peer then constructs an 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 shown. Some routes are cancelled due to a longer delay than $LastDelay$. Eventually, a good path $S_1 \rightarrow S_2 \rightarrow D$ is found. Then $LastDelay$ is updated. As a large portion of *ProM* messages are stopped during forwarding process, the overhead is acceptable.

A streaming path of Peer D is treated as invalid if (i) the source-to-end delay is larger than a given threshold $D(MS)$, or (ii) the direct parent of Peer D on the path leaves. In this design, we only disconnect the overlay link between the end peer to its parent node. There are two reasons: (i) the other connections on the path can be reserved to provide support for new incoming peers, and (ii) frequent disconnections incur a lot of unnecessary traffic.

- Managements of the active streaming path set

There are three operations, maintaining the states of active streaming paths, cutting off invalid paths, and adding new active paths from a backup set. When the total bit rates from active streaming paths are lower than $rate(MS)$, the manager will check whether a better path should be activated to replace the current one.

This manager has the following characteristics: (i) it employs a heuristic algorithm to optimized the system step by step; (ii) probing procedures are originated from the normal peers, not the source peer, so that the control overhead can be distributed; (iii) the number of forwarding neighbors, j , is chosen to balance the tradeoff between the effectiveness and the overhead; and (iv) the frequency of probing and optimization is dynamic. In IOO, the probing procedure is feedback-driven based on delay. Log data from AnySee, the prototype of IOO scheme to be discussed in Section 5.4, show that when peers are watching highly popular movies, they are willing to tolerate a larger delay. Hence, it is reasonable that different programs define different $D(MS)$.

3.5 Key Node Manager

Admission control is useful when there are too many requests. The goal of the key node manager is to determine the number of requests that a peer should have. Suppose a peer has Q unused connection slots. All requests will be classified into W types according to the different popularities of the target media, and each falls into one of W queues. When we assign the Q slots to W queues, there are two interesting cases. First, some queues are assigned with more than one connection slot, which can be modeled as an M/M/m/K queuing model. Second, some queues only receive one connection, which follows an M/M/1/K queuing model [25].

Our goal is as follows. Suppose the arriving rate of queue j is λ_j , and all arriving rates satisfy $\lambda_1 \leq \dots \leq \lambda_j \leq \dots \leq \lambda_w$. Assume the service rate to assign one connection slot is μ and each connection processor can buffer k requests ($k \geq 1$). If the probability that n requests follow the M/M/m/K queuing model is p_n , we have

$$p_n = \begin{cases} \frac{(m\rho)^n}{n!} p_0 & n = 0, 1 \dots m-1 \\ \frac{m^m \rho^n}{m!} p_0 & n = m, m+1 \dots K \end{cases} \quad (4)$$

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

$$p_0 = \begin{cases} \left[\sum_{i=0}^{m-1} \frac{(m\rho)^i}{i!} + \frac{(m\rho)^m}{m!} \frac{1-\rho^{K-m+1}}{1-\rho} \right]^{-1} & \rho \neq 1 \\ \left[\sum_{i=0}^{m-1} \frac{(m)^i}{i!} + \frac{(m)^m}{m!} (K-m+1) \right]^{-1} & \rho = 1 \end{cases} \quad (5)$$

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

$$\bar{\rho} = \rho(1 - \rho k) = \rho \left(1 - \frac{m^m \rho^k p_0}{m!} \right) \quad (6)$$

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

$$p'_n = \begin{cases} \frac{(1-\rho)\rho^n}{1-\rho^{K+1}} & \rho \neq 1 \\ \frac{1}{K+1} & \rho = 1 \end{cases} \quad 0 \leq n \leq K \bullet \quad (7)$$

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

$$\tilde{\rho} = 1 - p'_0 = \rho \left(\frac{1 - \rho^K}{1 - \rho^{K+1}} \right) \bullet \quad (8)$$

and p'_K is the failure probability of requests. Then, the target, resources utilization, can be expressed:

$$\text{Max}(\rho(Q_1, Q_2, \dots, Q_W)) = \text{Max} \left(\sum_{1 \leq i, j \leq W}^{i \neq j} (\tilde{\rho}_i + \tilde{\rho}_j) \right) \bullet \quad (9)$$

$$\text{Subject to } \sum_{i=1}^W Q_i = Q \quad 1 \leq Q_i < Q$$

Here, Q_i is the number of connection slots assigned to the request queue with ID i . The optimization problem in Eq. (9) can be divided into two parts. First, we enumerate all $(W, 1)$ -partitions (W queues and each should be allocated at least 1 connection slot) of Q spare connections such that the best allocation can be found to maximize $\rho(Q_1, Q_2, \dots, Q_W)$. Second, for all H partitions of Q slots, we compute all H results of average resource utilization and select the best partition. In the first phase, we can get H , the number of partitions of Q by

$$H = \binom{Q-1}{W-1} = \frac{(Q-1)!}{(W-1)!(Q-W)!}, \quad Q \geq W \quad (10)$$

From Equation (10), the complexity of the first algorithm is $O(Q)$. The second algorithm is to select the maximal one from H results with the complexity of $O(C_{W-1}^{Q-1}) = O(Q)$.

Overall, this optimization problem has complexity of $O(Q)$. Considering one normal peer with 10Mbps bandwidth and average streaming rate 300Kbps, Q should be set less than 33.

3.6 Buffer Manager

This manager is responsible for receiving valid media data from multiple providers in the active streaming path set and keeping the media playback. IOO employs a similar heuristic as used in the Coolstreaming system 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 Coolstreaming does not employ any inter-overlay optimization, peers often fail to find the closest neighbors to provide services. Consequently, to keep the media playback continuously, a big buffer must be used in Coolstreaming. With the help of IOO, a small buffer space is enough, which also means a shorter startup delay. For example, the average startup delay of AnySee system based on IOO scheme is about 10 seconds.

4 SIMULATION

Before introducing the implementation of IOO in the real AnySee system, we evaluate this design with simulations so that we can contrast its performance with a recent live streaming system, Coolstreaming. We mainly concern the metrics of *Resource utilization* and *Continuity index*.

4.1 Simulation Setup

We consider two types of topologies, physical topology and logical P2P topology. The physical topology should represent 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.

We have developed a crawler to collect topology information of Gnutella network [1]. According to Gnutella, a ping message with TTL=2 and HOP=0 is regarded as a crawler ping, and peers, upon receiving a crawler ping, would respond with appropriate pong messages. Based on this mechanism, we explore Gnutella topology by performing a breadth first search on the network. From our experience and observations, we find that some clients such as Gnucleus, Morpheus (based on GnucDNA), do not respond the crawler ping appropriately. Fortunately, those clients send an information page summarizing servants' status to any web browser trying to connect to it. Motivated by this, we also developed a web spider as a means of collecting topology information from these clients, and integrated the web spider into the crawler, which accelerates the crawling process significantly. The crawler is written in Java based on Limewire's [3] open source client, and runs in parallel using 40 threads. Our crawler can discover more than 50,000 peers within half an hour. In this simulation we use three data sets, obtained from different time slots. For the physical topology, we use BRITE [4] generating three topologies, each has 5,000 nodes.

The major parameters in our simulations are listed in Table 1. In each run, peers randomly join one of S streaming overlays ($S=1, 4, 8, 12$). A peer has bandwidth BW ranging from 1 to 10 Mbps and maintains M neighbors. The size of an 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 $D(MS)$ is set to 25 seconds, which is estimated from Logs of our AnySee implementation. The adjustment factor p is set to 1, which means we provide one backup streaming path for each active streaming path.

Table 1 Simulation Parameters

Abbreviate	Comment
S	number of streaming overlays
M	number of neighbors
N	size of one overlay
r	streaming playback rate
BW	total bandwidth in Mbps

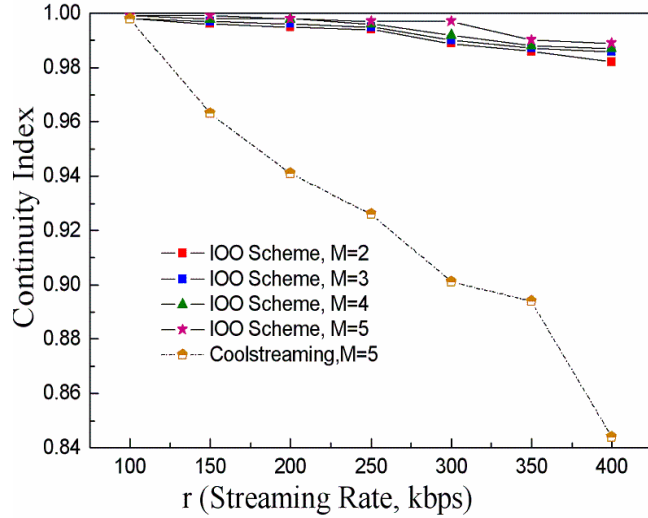


Fig. 7. Continuity index V.S. streaming rates when $N=400$, $S=12$ and initial buffer size is 40 seconds.

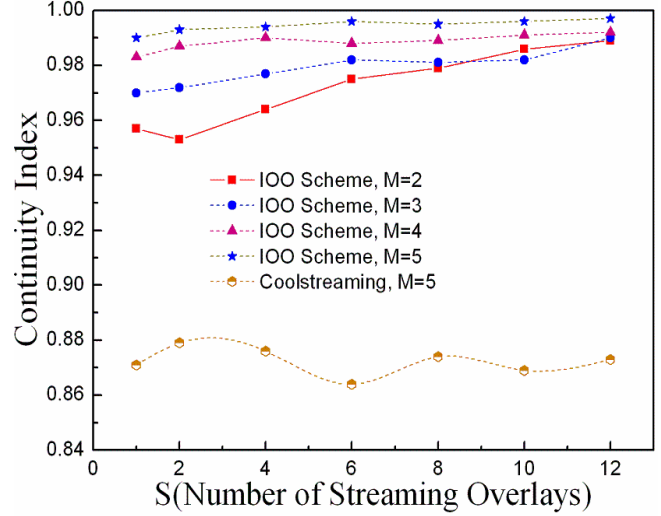


Fig. 8. Continuity index V.S. number of streaming overlays when $N=400$, $r=300$ Kbps and initial buffer size is 40 seconds.

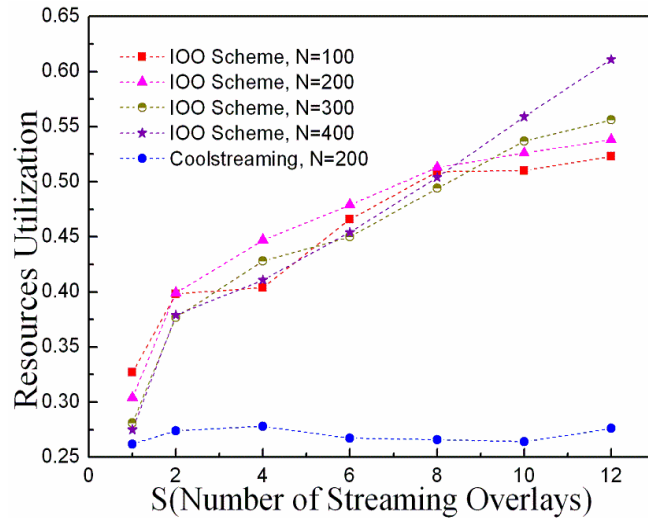


Fig. 9. Resources utilization: overlay size V.S. the number of streaming overlays when $M=12$, $r=300$ Kbps.

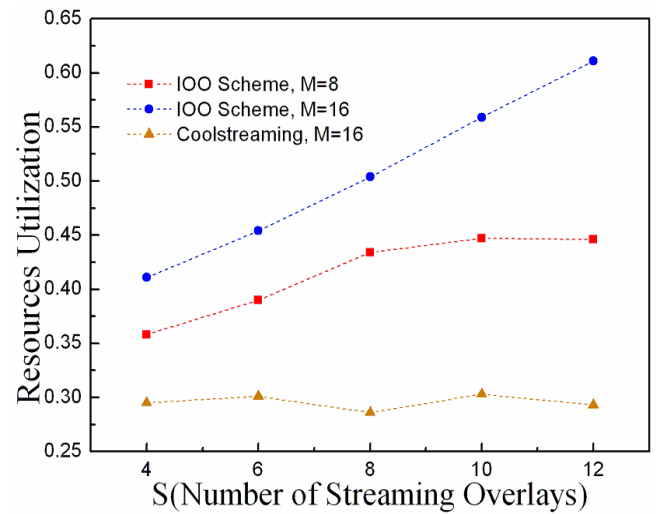


Fig. 10. Resources utilization: the number of neighbors V.S. the number of streaming overlays when $N=400$, $r=300$ Kbps.

4.2 Simulation Results

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

Figure 7 plots the continuity index against streaming rate, where we contrast IOO with Coolstreaming. When the streaming rate is increased, the continuity of IOO scheme is not changed much while the continuity of Coolstreaming is degraded. There are two reasons. First, IOO scheme can find neighbors from all peering nodes to request services, while Coolstreaming is only able to find suppliers from the same overlay. Second, the necessary buffer size of IOO scheme is 40 seconds, while Cool-

streaming often needs a 120-second buffer.

Figure 8 shows the scalability of IOO. We can see that the continuity improves with the increasing number of overlays. Our implementation results also show that the larger overlay size often leads to better QoS. From Fig. 8 we also see that IOO scheme outperforms Coolstreaming, even they have the same number of overlays.

Figures 9 and Figure 10 contrast resource utilization. As seen in Fig. 9, a larger number of overlays have a greater impact on the performance of IOO scheme, but no obvious influence on that of Coolstreaming. In Fig. 10, the resource utilization is improved with the increasing number of neighbors. Under the same condition, the performance of IOO scheme is better than that of Coolstreaming.

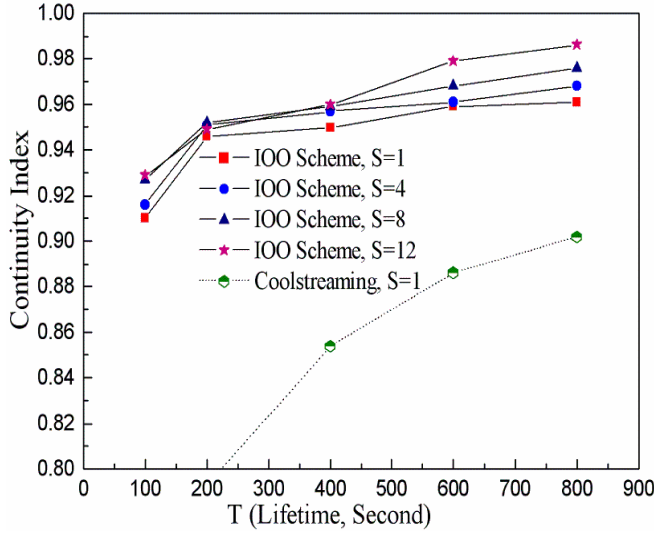


Fig. 11. Continuity index under dynamic environment when $M=5$, $N=400$, $r=300$ Kbps and initial buffer size is 40 seconds.

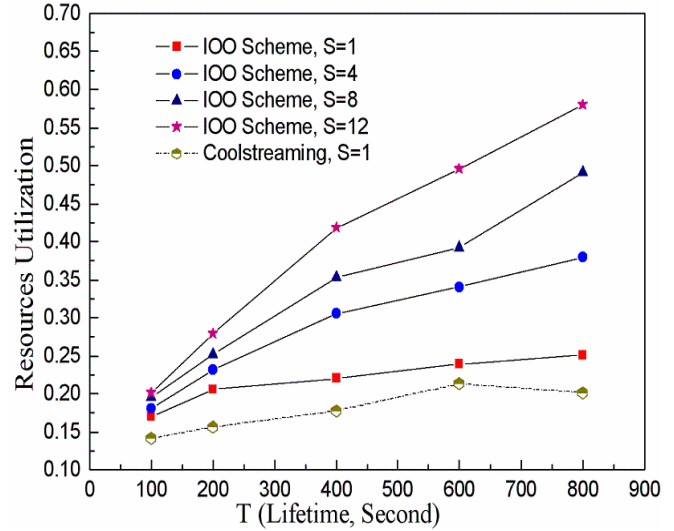


Fig. 12. Resources utilization under dynamic environment when $M=5$, $N=400$, and $r=300$ Kbps.

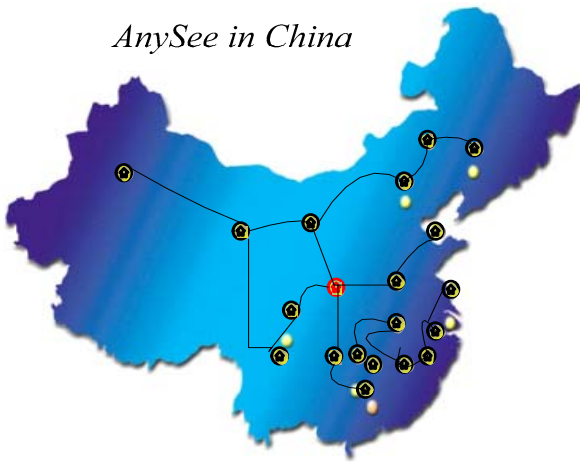


Fig. 13. Service map of AnySee in CERNET of China (red center point: HUST, Wuhan).

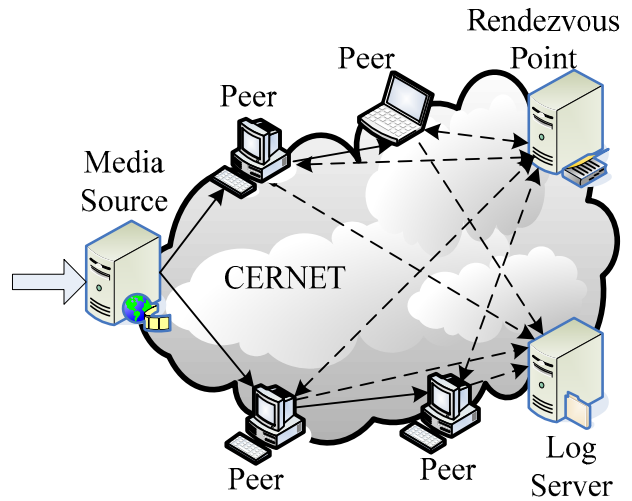


Fig. 14. Topology of AnySee in CERNET of China.

We then allow peers leaving and joining freely. We define the lifetime of each peer in the overlay from 100 seconds to 800 seconds. Peer average lifetime is exponentially distributed with an average of T seconds. We can see from Figures 11 and 12 that longer lifetime leads to better service quality and higher resource utilization. As our proposed IOO scheme has a backup path management design and the reverse tracing component keeps finding better paths dynamically, IOO scheme always outperforms Coolstreaming.

5 ANYSEE: PRACTICAL IMPLEMENTATION OF IOO SCHEME

We have implemented the public free system, AnySee, and released several versions (v.1.1, v2.0 and v3.0) to provide a scalable live-streaming service platform based on inter-overlay optimization in CERNET of China. CERNET is

funded by the Chinese government to support network services for people in universities. Now more than 15 million PCs and 26 million users have been connected to CERNET to share information. From June 2004 to February 2005, there were over 60,000 connections to AnySee platform and over 40 universities and 20 cities in China were in the service map as illustrated in Fig. 13. AnySee² is implemented with Java and is platform-independent.

5.1 Architecture overview

AnySee is comprised of four major components as illustrated in Fig. 14. They are (i) a rendezvous point, (ii) a media source, (iii) a log server, and (iv) peers.

²AnySee system has three main versions up to March 2007. The version we discussed is 1.1. The updated version 3.0 is published on the web site of www.anysee.net, which is in CERNET of China. The latest version adds time-shift functions.

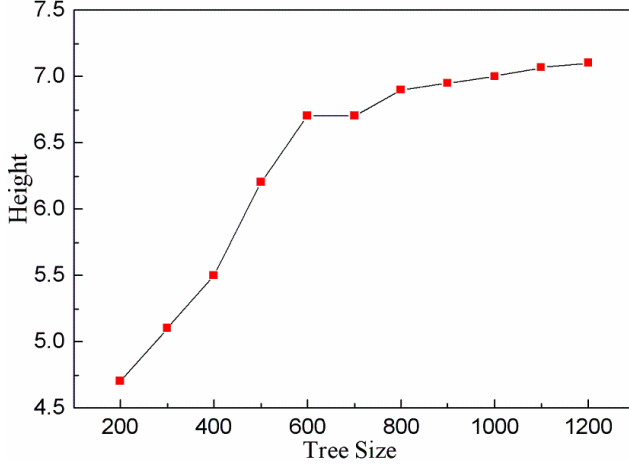


Fig. 15. Tree height V.S. size.

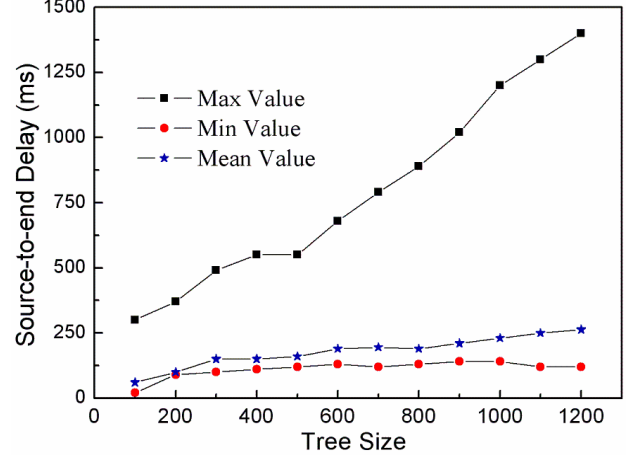


Fig. 16. Source-to-end delay V.S. tree size.

Each peer contains several important modules: an IP to network coordinates database, which is pre-built and integrated into the peer's software, for optimizing the underlying mesh-based overlay according to GID scheme; the topology manager, including underlying mesh manager, tree manager and inter-overlay manager; a buffer manager; a media player, decoding streaming data and local HTTP/RTSP server, receiving media data from buffer, and sending them to media player using HTTP or RTSP protocol.

Normally, bandwidths of up-stream and down-stream for each peer in CERNET are 10Mbps to 100Mbps. One peer can provide streaming services for multiple ones in AnySee. In our log, we can see each peer is often able to maintain 5 neighbors with the streaming rate from 300Kbps to 550Kbps, which means it joins 5 streaming overlays simultaneously.

5.2 Implementation Experiences

We discuss two interesting issues in AnySee implementation, GID based scheme and locality-aware buffer management scheme, which are not included in IOO.

• GID based scheme

Low latency and high bandwidth are always desirable in streaming services. Due to the fact that all peers in AnySee are in the same CERNET so the physical network map is known, the distance among pairs of peers can be easily computed by the IP addresses. Hence, AnySee requires each peer maintain a network coordinate database (INCD), such that every peer has a position, named. 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. With the help of the GID based scheme, a peer can find closer peers to build connections instead of randomly selecting neighbors. Consequently, LTM (described in section 3.2) needs less management resources than ever to build mesh-based.

• Locality-aware buffer management scheme

In IOO scheme, the buffer size is simply defined as 40 seconds. As the behavior of peers in upper layers of a tree often have larger impact on QoS than that of peers in lower

layers, AnySee employs a layer-aware buffer management mechanism. Each peer computes its appropriate buffer space size based on its layer number. In AnySee, the size of buffer for peer A at the x-th layer is given by:

$$T_A = f(x) = \varepsilon \times TL_A + TD_A \bullet \quad (11)$$

where TL_A denotes the total link delay, TD_A is the total transmission 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 shift delay, the average time to replace the removed parent with a new one, is $t_l = P_b \times t_b$ and the link delay TL_A is the accumulation of all shift delays. Suppose the transmission delay per hop is γ and the total hops between the source peer and peer A is x , the total transmission delay of peer A is $TD_A = \gamma \times x$. If the path from source peer to peer A is $l_{S \rightarrow A} = \{l_{S \rightarrow a_1}, l_{a_1 \rightarrow a_2}, \dots, l_{a_{m-1} \rightarrow A}\}$, the path has the following properties: (i) the source peer would persist all the time and the path $l_{S \rightarrow a_1}$ would not break; (ii) peer A would also stay in the network and the path $l_{a_{m-1} \rightarrow A}$ would exist; (iii) the influence that multiple borders break down simultaneously is the accumulation of influence that multiple borders break down one by one; (iv) if peer a_i leaves, a new peer would join the tree and replace the position of a_i . The total link delay can be computed by

$$TL_A = \sum_{i=1}^{i=x-1} t_{l_{a_i \rightarrow a_{i+1}}}$$

and

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

then,

$$TL_A = t_l + (1 - P_b) \times t_l + \dots + (1 - P_b)^{x-2} \times t_l$$

$$TL_A = t_l \times \frac{1 - (1 - P_b)^{x-1}}{P_b} = t_b \times \left(1 - (1 - P_b)^{x-1}\right) \bullet \bullet \quad (12)$$

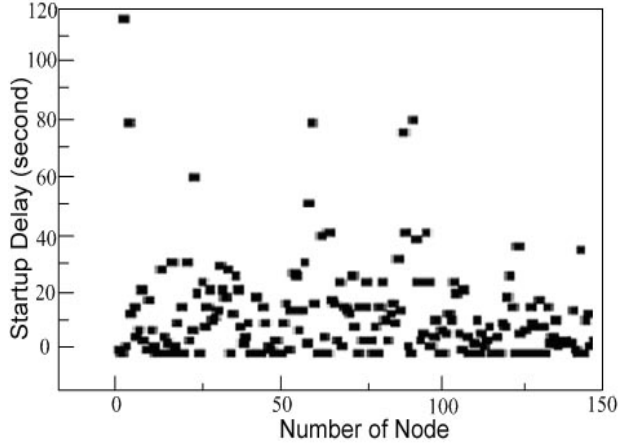


Fig. 17. Startup delay in AnySee.

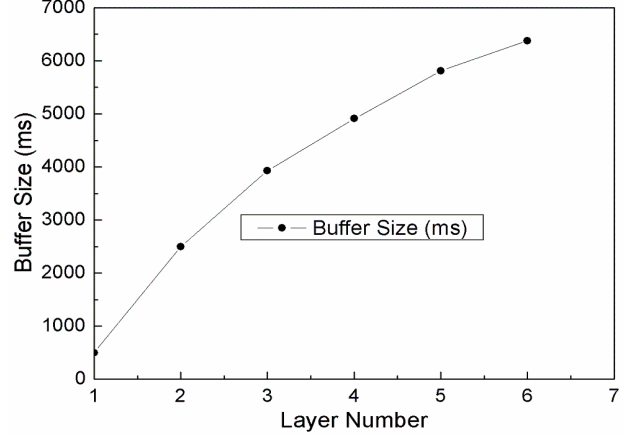


Fig. 18. Buffer size V.S. layer number.

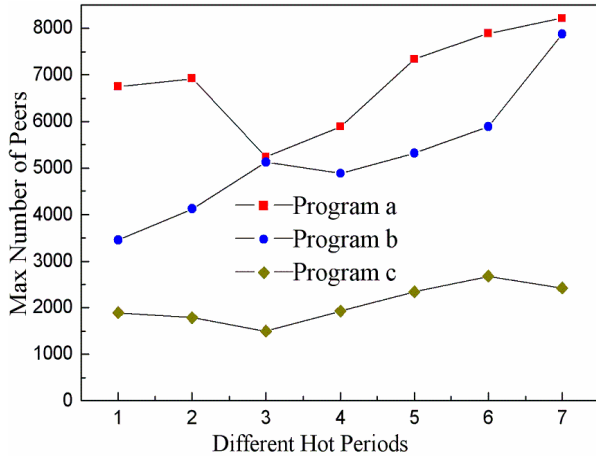


Fig.19 Number of peers.

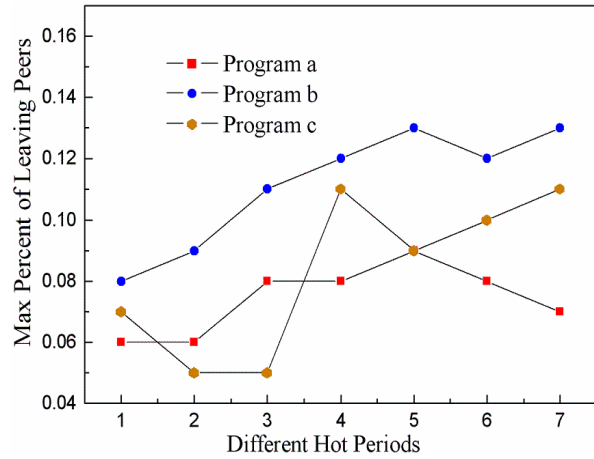


Fig.20 Percentage of leaving peers.

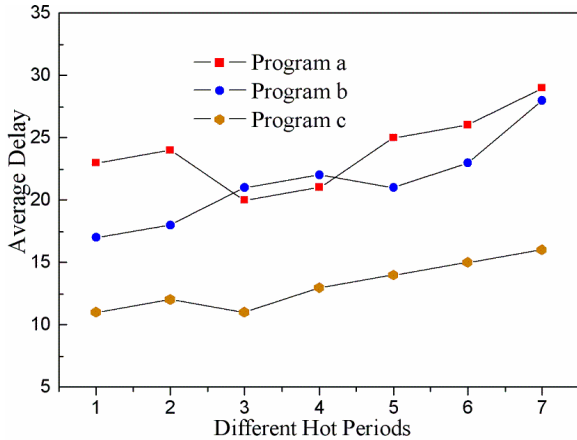


Fig. 21. Average delay.

Thus we have

$$T_A = \varepsilon \times t_b \times (1 - (1 - P_b)^{x-1}) + \gamma \times x \quad (13)$$

Given the estimation of the above parameters in Eq. (13), the maximum buffer size to offset dynamic factors for A at

the x -th layer is computed based on the layer number x .

5.3 Performance

We select to show the log data from 13/08/2004 to 29/08/2004, during which period 7200 users from 40 universities in 14 cities of China received streaming services from AnySee. We analyze the performance of the multiple multicast trees every ten minutes.

Figure 15 plots the average height of AnySee trees against tree size. The height increases when more peers join, but it is always less than 7 even up to a thousand peers are included in one tree. Such a property significantly shortens the source-to-end delay, shown in Fig.16. We can see that the maximal value of source-to-end delay increases with the incensement of tree size, while the average delay is typically at the level of around 200ms.

Figure 17 plots the startup delay. This delay of most peers is less than 20 seconds. We have also implemented a simple prototype, getting media services from Coolstreaming network, and we observe the startup delays for 50 times. Mostly, the startup delay of Coolstreaming is around 60 seconds.

Table 2 ADLs 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

Based on the results shown in Fig. 16, we set γ to 200 ms and define $t_b = 350$, $P_b = 0.4$, and $\varepsilon = 10$. Figure 18 shows the corresponding values of buffer sizes in different layers. The maximal number of layers is not exceeding 8. When the buffer sizes are set based on the values in Fig. 18, AnySee is not influenced much by peer frequent joining and leaving.

5.4 User behaviors

Intuitively, if we know more about the user behaviors, we can better optimize the system. By analyzing the log data, we try to get man parameters, including the total number of peers, the average delay (source-to-end delay), and leaving peer percentage in different "hot" periods.

Figures 19, 20 and 21 show the maximum number of peers, maximum percentage of leaving peers, and average delay of three programs, a, b and c for one hour. The results show that the overlays with popular movies attract more users to join and stay, but cause large 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 service. For example, the leaving percentage of Program-a is not the largest while its average delay is the longest. Peers have more patience if the program is popular and attractive. Second, delays from 20 to 30 seconds will not be the killer for the live streaming services. Most people will stay in the overlay even if there is a 30 second delay from the source peer.

Based on the above observations, AnySee determines the optimization frequency according to the average delay together with the percentage of leaving peers. The parameter "optimization index", ADL, is given by

$$ADL = 100 \times \frac{\text{leaving percent}}{\text{average delay}} \quad (14)$$

Three ADLs from different periods are shown in Table 2. From Table 2, the average ADLs for the above programs are 0.5937, 0.5213 and 0.6215, respectively. AnySee uses a threshold on ADL to determine whether an optimization operation is needed.

6 CONCLUSION AND FUTURE WORK

Efficient and scalable live-streaming overlay construction is a hot topic. In this paper, we propose an inter-overlay optimization based live streaming scheme, IOO. Instead of selecting better paths in the same overlay, IOO constructs

efficient paths using peers in different overlays. We evaluate the performance of IOO scheme by comprehensive simulations. Our experimental results show that IOO scheme outperforms existing intra-overlay live streaming schemes. In the future research, we will provide a reputation based incentive mechanism encouraging users to contribute resources.

The practical AnySee system, a prototype of IOO scheme, 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, GCC'04 and NPC'04, and other massive entertainment programs. Logs from AnySee show that users have great patience for live streaming services with relatively large delay if they have enough interest on the programs. We hope the AnySee system can serve more people and attain better quality in the future.

We are also building a large-scale P2P VOD service. We are going to observe more user behaviors to further improve the system performance.

ACKNOWLEDGMENTS

This work was supported in part by China National Natural Science Foundation (NSFC) Grants No. 60642010, 60433040, the Research Fund for the Doctoral Program of Higher Education Grants No. 20050487040, and Hong Kong RGC Grants HKUST6264/04E and HKUST 6152/06E. The preliminary result of this paper was published in Proceedings of IEEE INFOCOM 2006.

REFERENCES

- [1] The Gnutella protocol specification 0.6, <http://rfc-gnutella.sourceforge.net>.
- [2] AnySee, a live streaming peer-to-peer platform, <http://grid.hust.edu.cn/p2p>.
- [3] Limewire, <http://www.limewire.com>.
- [4] BRITE, <http://www.cs.bu.edu/brite>.
- [5] 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.
- [6] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The Feasibility of Peer-To-Peer Architectures for Large-Scale Live Streaming Application", In Proceedings of ACM SIGCOMM, 2004.
- [7] Q. Zhang, F. Yang, W. Zhu, and Y.-Q. Zhang, "A Construction of Locality-Aware Overlay Network: mOverlay and Its Performance", IEEE JSAC Special Issue on Recent Advances on Service Overlay Networks, Jan. 2004.

[8] 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.

[9] Y. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast", In Proceedings of *ACM SIGMETRICS*, 2000.

[10] 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.

[11] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using An Overlay Mesh", In Proceedings of *ACM SOSP*, 2003.

[12] 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.

[13] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure", In Proceedings of *Usenix Symposium on Internet Technologies and Systems*, 2001.

[14] B. Zhang, S. Jamin, and L. Zhang, "Host Multicast: A Framework for Delivering Multicast to End Users", In Proceedings of *IEEE INFOCOM*, 2002.

[15] M. Hefeeda, A. Habib, B. Botev et al, "PROMISE: Peer-to-Peer Media Streaming Using Collectcast", In Proceedings of *ACM Multimedia*, 2003.

[16] D. Tran, K. Hua and S. Sheu, "ZIGZAG: An Efficient Peer-To-Peer Scheme for VOD Service", In Proceedings of *WWW*, 2003.

[17] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast", In Proceedings of *ACM SIGCOMM*, 2002.

[18] Y. Guo, K. Suh, J. Kurose, D. Towsley, "P2Cast: P2P Patching Scheme for VOD Service", In Proceedings of *WWW*, 2003.

[19] X. Jiang, Y. Dong, X. D, B. Bhargava, "GNUSTREAM: A P2P Media Streaming System Prototype", In Proceedings of *IEEE ICME*, 2003.

[20] Z. Zhang, S. Shi, and J. Zhu, "SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT", In Proceedings of *IEEE IPTPS*, 2003.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network", In Proceedings of *ACM SIGCOMM*, 2001.

[22] Y. Liu, L. Xiao, X. Liu, L.M. Ni, and X. Zhang, "Location-Aware Topology Matching in P2P Systems", In Proceedings of *IEEE INFOCOM*, 2004.

[23] M. Ripeanu and I. Foster, "Mapping Gnutella Network", *IEEE Internet Computing*, 2002.

[24] NTP: The Network Time Protocol, <http://www.ntp.org/>.

[25] L. Kleinrock, *Queueing Systems*, John Wiley, 1974.

[26] X. Tu, H. Jin, D. Deng, C. Zhang, Q. Yuan, "Design and Deployment of Locality-Aware Overlay Multicast Protocol for Live Streaming Services", In Proceedings of *NPC*, 2005.



Xiaofei Liao received a Ph.D degree in computer science and engineering from Huazhong University of Science and Technology (HUST), China, in 2005. He is now an associate professor in school of Computer Science and Engineering at HUST. His research interests are in the areas of peer-to-peer system, cluster computing and streaming services. He is a member of the IEEE and the IEEE Computer Society.



Hai Jin received a BS, a MA and a Ph.D. degree in computer engineering from Huazhong University of Science and Technology (HUST) in 1988, 1991 and 1994, respectively. Now He is a Professor of Computer Science and Engineering at HUST in China. He is now the Dean of School of Computer Science and Technology at HUST. In 1996, he was awarded German Academic Exchange Service (DAAD) fellowship for visiting the Technical University of Chemnitz in Germany. He worked for the University of Hong Kong between 1998 and 2000 and participated in the HKU Cluster project. He worked as a visiting scholar at the

University of Southern California between 1999 and 2000. He is the chief scientist of the largest grid computing project, ChinaGrid, in China. Dr. Jin is a senior member of IEEE and member of ACM. He is the member of Grid Forum Steering Group (GFSG). His research interests include cluster computing and grid computing, peer-to-peer computing, network storage, network security, and high assurance computing.



Yunhao Liu received his BS degree in Automation Department from Tsinghua University, China, in 1995, and an MA degree in Beijing Foreign Studies University, China, in 1997, and an MS and a Ph.D. degree in Computer Science and Engineering at Michigan State University in 2003 and 2004, respectively. He is now an assistant professor in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. His research interests include peer-to-peer computing, sensor networks, and pervasive computing. He is a senior member of the IEEE and the IEEE Computer Society.



Lionel M. Ni earned his Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, in 1980. He is Chair Professor at the Hong Kong University of Science and Technology and heads the Computer Science and Engineering Department. He is also Director of the HKUST China Ministry of Education/Microsoft Research Asia IT Key Lab and Director of HKUST Digital Life Research Center.

A fellow of IEEE, Dr. Ni has chaired many professional conferences and has received a number of awards for authoring outstanding papers. He is a co-author of the book "Interconnection Networks: An Engineering Approach" (with Jose Duato and Sudhakar Yalamanchili) published by Morgan Kaufmann in 2002, the book "Smart Phone and Next Generation Mobile Computing" (with Pei Zheng) published by Morgan Kaufmann in 2006, and the book "Professional Smartphone Programming" (with Baijian Yang and Pei Zheng) published by Wrox Publishing Inc. in 2007.