Agiler: A P2P Live Streaming System with Low Playback Lag

Dongbo Huang, Jin Zhao and Xin Wang
School of Computer Science, Fudan University, China
Shanghai Key Lab of Intelligent Information Processing, Shanghai, China
Email: {082024082, jzhao, xinw}@fudan.edu.cn

Abstract—**Short playback lag is preferred in many urgent and interactive scenarios such as live sports and distance education. However, measurement studies have shown that many popular P2P live streaming systems still suffer from long playback lag, say, more than 100 seconds, which makes the live streaming less realtime. Due to the unstructured nature of P2P networks, it is really a challenge to reduce the playback lag in P2P live streaming system under limited server bandwidth. In this paper, we propose a novel P2P live streaming system - Agiler that aims at reducing the playback lag in lag-sensitive applications under limited server bandwidth. In Agiler, we first group the peers into clusters according to the Autonomous System Number (ASN) and then spread the broadcast in ripples that gradually increase in playback lag as we move away from the media server. It adopts synchronous playback within a cluster to accelerate the chunk diffusion and asynchronous playback among different clusters to facilitate the chunk swarming. Combined with the partitioned buffer strategy and the proportional playback lag strategy, the newly generated fresh chunks can be delivered to all peers in time. The proposed system is evaluated through extensive packet-level simulations, which show the bandwidth utilization ratio of the peers is improved and the average playback lag is significantly reduced.**

I. Introduction

In recent years, both the academia and industry have shown strong interest in P2P streaming systems. P2P streaming systems have become one of the most popular technologies for distributing streaming videos to end users, which has been proved by the successful deployments of commercial P2P streaming systems such as PPLive [1] and PPStream [2]. Only with a dedicated streaming server of limited bandwidth like several Mbps, are P2P streaming systems able to provide video service to thousands of users at the same time. For example, the PPLive system has had more than $100,000$ simultaneously online users for a live broadcast of a popular TV program [3].

Nowadays, mesh-pull architecture is widely adopted by many real-deployed P2P streaming systems such as the systems mentioned above according to [3, 4]. In the mesh-pull architecture, a live video is divided into media chunks at an original media server and then these media chunks are injected into the system (actually the chunks are requested by the average peers). The mesh-pull architecture lacks of a structured overlay and the peers communicate with each other using gossip-like protocols. Each peer exchanges chunk availability information periodically (i.e., once per Δt) with its neighbors and determines which chunk is to be requested from which neighbor accordingly. The mesh-pull architecture has several advantages as follows: (i) overlay construction and maintenance are very simple, (ii) each peer are more likely to have diverse paths which in turn reduces the probability of chunk shortage, the resulting overlay is very resilient to churn. (iii) the outgoing bandwidth of most participating peers is effectively utilized, it is self-scaling.

We shall first explain several important concepts before we go a step further. *Source-to-end delay* is the time needed to transfer the media chunk generated by the media server to a certain peer, which is mentioned in many previous literature. *Playback lag* of a peer refers to the interval from a chunk is generated at the media server to the moment it is played at the peer. *Start-up latency* is the time interval from when one channel is selected until actual playback starts on the screen. The start-up latency and the playback lag are two very important metrics for the user experience and this paper lays stress on the playback lag in P2P live streaming systems.

Learned from the measurement study [3], the playback lag in PPLive [1], one of the most popular P2P streaming systems in the world, is more than 100 seconds. The large playback lag makes no difference when users are watching a channel without lag requirement, such as pre-recorded TV shows, movies. Indeed, most real-world P2P streaming systems work in this fashion, having users experiencing minutes of playback lag. However, there are a series of live channels that require short playback lag. Fox example, live sports and distance education that are broadcast live. In these scenarios, long playback lag definitely has a bad influence on user experience. Increasing the server bandwidth definitely could shorten the playback lag, but the server bandwidth is limited and costly. Therefore, it is urgent for us to reduce the playback lag under limited server bandwidth.

In this paper, we seek to design a more realtime P2P live streaming system with low playback lag for lag-sensitive applications. It is a challenge due to following two important reasons: (i) the limited availability of future media chunks in live streaming applications, and (ii) the media chunks must arrive the peer before their playback deadlines. Overlay topology construction, chunk scheduling algorithms and playback lag setting strategy are the three important factors influencing the in-time arrivals of fresh media chunks. Therefore we start from these three aspects when designing this new system.

We propose a novel asynchronous playback P2P live streaming system - Agiler, in which peers within an overlay (peers who are watching the same video or channel) are grouped into clusters based on the ASN. The peers in the same cluster is synchronized in playback and the playback lag of a cluster is in proportion to the network distance between the cluster and the media server (actually the playback lag is calculated by the first few online peers of a cluster). Combined with our special

COLLABORATECOM 2010, October 9-12, Chicago, USA
Copyright © 2011 ICST
DOI 10.4108/icst.collaboratecom.2010.13

partitioned buffer strategy, Agiler can provide good streaming quality with low playback lag. As our design, analysis and simulation studies have illustrated, Agiler is able to take full advantage of a limited pool of server bandwidth to achieve shorter playback lag without degradation of the performance compared with synchronized strategies.

To sum up, the primary contributions of this paper are as follows. We propose Agiler, a more real-time P2P live streaming system that reduce the average playback lag by more than 20% under limited server bandwidth. And the ratio is up to 40% for the superior peers near the media server. AS-based clustering strategy is verified to be an effective way to construct an optimal overlay topology that can reduce the playback lag. The proportional playback lag strategy and partitioned buffer strategy are brought forward and evaluated by simulations in Agiler. In addition, biased neighbor selection algorithm is combined with our proportional playback lag strategy in Agiler, which can decrease the transmission delay and bring down the costly cross-ISP (Internet Service Provider) traffic.

The remainder of this paper is organized as follows. In sec. II, we discuss the originality of our work in the context of related work. The detailed design of our system is presented in Sec. III. In Sec. IV, we present our results from a series of simulations to demonstrate the efficiency of our design. We conclude the paper and describe our future plans in Sec. V.

II. Related Work

As a matter of fact, there are many measurement studies of real-world P2P live streaming systems showing that large playback lag exists in popular P2P live streaming systems. Ali et al. [5] analyze the performance and characteristics of the most popular P2P live streaming systems - PPLive and SOPCast, which is the first one of that kind of studies. This study presents a framework to analyze P2P applications from one single point and then analyze the probable operation mode, resource usage, locality and stability of data in P2P live streaming systems. It is worth noticing that there is no locality-awareness when a peer selects neighbors, which cause unnecessary transmission cost and inefficiency of the system. X. Hei et al. [3] build a buffer map crawler and deploy passive sniffing nodes to study the performance and characteristics of PPLive, one of the most popular P2P streaming systems. This study find that users in the measured P2P streaming system still suffer from long start-up latency and large playback lag, ranging from several seconds to a couple of minutes.

Many existing works on P2P live streaming systems focus on overlay construction [6], scheduling of media chunks [7], incentives [8]. Other works exploit the coding techniques such as network coding [9], multiple description coding [10] and scalable video coding [11] to simplify the scheduling and enhance the resilience of the system. They try every mean to improve the streaming quality and scalability of the system but ignore the long playback lag. Consequently, there are few works addressing the playback lag in P2P live streaming systems, which is of great importance for lag-sensitive applications.

There are some theoretical studies exploring the delay issues in P2P live streaming systems. D. Ren et al. [12] design an overlay which achieves low source-to-end delay. It accommodates the asymmetric and diverse uplink bandwidth and is robust to peer dynamics. They first formulate the minimum delay mesh problem and show that it is NP-hard. Then they propose a centralized heuristic algorithm based on complete knowledge to minimize source-to-end delay. However, some parameters needed in the centralized heuristic algorithm is hardly to obtain, so it is unpractical to use this heuristic algorithm to reduce the delay in the mesh-based P2P live streaming systems. Besides, low source-to-end delay does not mean short playback lag. Other strategies are required to effectively reduce the playback lag.

Y. Liu theoretically studies the impact of the inherent delay constraint and derive the minimum delay bounds for real time P2P streaming systems based on a snow-ball streaming algorithm in [13], which is inspirational for proposing good solutions to reduce delays in P2P live streaming systems. Performance gap between the fundamental limits and the actual performance of mesh-pull protocol has been mathematically analyzed using a unified framework based on trellis graph techniques [14], in which source-to-end delay and start-up latency is studied.

iGridMedia, proposed by Zhang et al. [15], investigates the relationships between playback lag guarantee and the consumption of server bandwidth. Both R^2 and iGridMedia adopt the synchronized playback algorithm in which all peers play the same media chunk simultaneously. In more recent works, D. Wu et al. [16] propose a radically different cross-channel P2P streaming framework, called View-Upload Decoupling (VUD). VUD strictly decouples peer downloading from uploading, which could bring stability to multichannel systems, enable cross-channel resource sharing and achieve lower switching delay and playback lag. However, VUD is in its juvenility so there is much work to be done to turn VUD into a real system that could be wildly deployed.

A full implementation of live P2P streaming system with improved playback called SonicStream is presented in [17]. With the use of network coding, SonicStream can improve the chunk availability and decrease the frequency of buffer map exchange, leading to the improved playback lag. Our previous work [18], builds a 3-level, hierarchical overlay where the peers are grouped according to their degrees of activity. Consequently, an optimal overlay construction is built and superior peers who contribute more are guaranteed with low playback lag. In this paper, we seek to propose a P2P live streaming system with low playback lag from a different perspective. The clustering strategy based on ASN in overlay construction, biased chunk scheduling and proportional playback lag strategy are designed with elaboration to reduce playback lag.

III. System Design

A. Problem Description

In Fig. 1, we illustrate the segment dissemination in mesh-pull based P2P live streaming systems. A media segment consists of several consecutive media chunks. The segment dissemination can be interpreted as a two-phase process. Diffusion phase: first different media chunks of the segment are rapidly delivered to a different subset of peers, as shown by the straight arrows in Fig. 1. Swarming phase: participating peers exchange their media chunks until each peer has a proper number of media chunks for the segment, as shown by the curly arrows in Fig. 1. Every rectangle marked with dotted lines in Fig. 1 represents a diffusion subtree. At the end of the diffusion phase of a segment, all peers in the overlay have at least one media chunk of the segment. During the swarming phase of a segment, participating peers pull the missing media chunks of the segment from other diffusion subtrees.

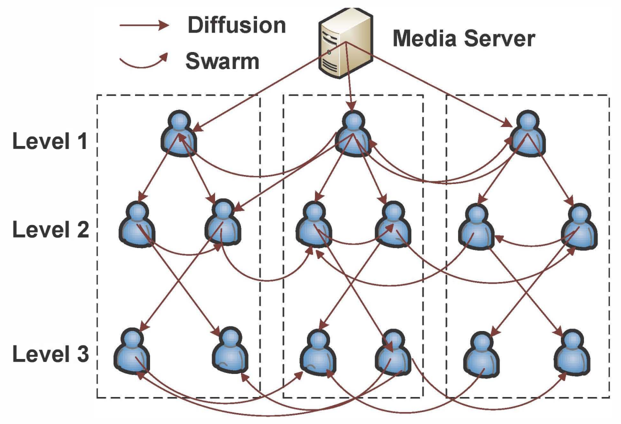

Fig. 1. Chunk dissemination in a mesh-pull overlay. Some connections are not shown for clarity of the figure.

Now let's take a look at the chunk diffusion delay in a diffusion subtree that is rooted in a peer at level 1. Without loss of generality, we assume the chunk size is one, and choose the streaming rate as the bandwidth unit. Accordingly, the chunk transmission time on a unit bandwidth link is 1 time slot, which equals to the chunk playback time. For the convenience of analysis, let's assume that the propagation delay between two any peers is dominated by the chunk transmission delay, thus it can be ignored. Based on the above assumption, we can get the following theorem according to [13]. If peers in a diffusion subtree form a M-level hierarchy with $\prod_{k=1}^{i} N_k$ peers on level i with uploading capacity of C_i,$(C_i > C_{i+1} \geqslant 1)$, there exits a continuous streaming schedule such that chunks can be streamed to all peers with a delay of $\eta = M + \sum_{j=1}^{M} \frac{\lceil log_2(N_i(C_i - 1)) \rceil}{C_i}$, where $C_0 = 2$. In a P2P live streaming system the following condition should be satisfied $(\eta + \phi) \leq \omega$, where ϕ denotes the swarming interval and ω represents the playback lag.

In order to reduce the playback lag in P2P live streaming systems, intuitively, we should minimize the delay for delivery of a segment. Based on the analysis above, we make efforts from the following two aspects. On one hand, we construct a more "flatten" overlay, which could reduce the number of hops for a chunk to reach all peers in a diffusion subtree. Increasing the degree of peers and selecting the "nearest" peers as neighbors contribute to constructing a "flatten" overlay. It would take less time for a chunk to be transmitted to peers in the same AS than peers from other ASes. On the other hand, the proportional playback lag setting among different clusters and caching media chunks that have been played can reduce the swarming interval. We set the playback lags of different clusters according to the network distance between the cluster and the media server. In doing so, the chunks have been played by peer in the top levels are reserved and relayed to the peers in other diffusion subtrees, which facilitate the chunk swarming.

B. System Overview

The system architecture of Agiler is showed in Fig. 2. It uses the mesh-pull architecture which have been adopted by many real-world commercial P2P streaming systems. The newly joined Agiler peer (we call it a newcomer) contacts the locality server to query the ASN and other geographic information about itself if the newcomer runs for the first time or detects that the network ID has changed. Similar to many other P2P streaming applications, the newcomer retrieves channel list or just update it from the channel server. Then it registers itself to a tracker and downloads the meta data for the channel that has been selected to watch. The tracker replies with a list of initial peers who are watching the same video. Afterwards, the newcomer contacts some of the initial peers according to certain rules to get the chunk availability information and request media chunks from them. The media server for this channel could be regarded as a special peer and is likely appear in the initial peer list.

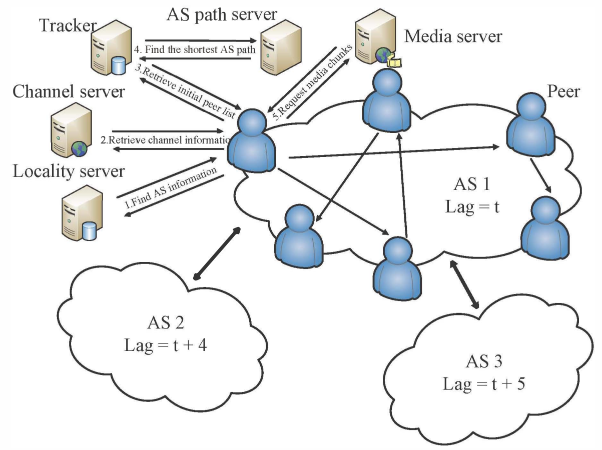

Fig. 2. System architecture

Just like many typical P2P streaming systems, there are several servers in Agiler working together to guarantee the smooth running of the system. The locality server is usually not deployed in P2P systems, which is responsible for locality information queries. The locality server does not need to be deployed by ourselves because Prefix WhoIs [19] can do this job for us. Prefix WhoIs uses the Internet's global routing table as gleaned from a number of routing peers around the world to disclose various up-to-date routing information, such as ASN, latitude, longitude and country name. Furthermore, it provides server software, client software and software development libraries for public use, so we could retrieve peer locality information from Prefix WhoIs database. The channel server is in charge of distributing the latest channel lists to newcomer, which include the channel information such as channel title, streaming rate, IP and port number.

The AS path server is a specific component of Agiler, one of which can serve many trackers. It is in charge of calculating the k (usually 2) shortest path from the source AS (the AS that the media server located in) to the destination AS (the AS that the newly joined peer located in). We can use the AS relationships provided by CAIDA [20] to calculate these AS paths. Each file from inferred AS relationships dataset of CAIDA contains a full AS graph derived from BGP table snapshots taken at 8-hour intervals over a 5-day period. The AS relationships available are customer-provider (and provider-customer in the opposite direction), peer-to-peer, and sibling-to-sibling.

Dijkstra algorithm could be used to find the shortest path and the second shortest path from source AS to destination AS. We also can employ the algorithms [21] proposed by David Eppstein to find the k shortest paths. The algorithms he proposed output an implicit representation of these paths in a digraph with n vertices and m edges in time $O(m+nlogn+k)$. Moreover, the algorithms find the k shortest paths from a given source s to each vertex in the graph in total time $O(m + nlogn + kn)$. It is worthy to notice that there already exist several implementations of these algorithms. We can find the k closest upstream neighbor ASes if we have got the k shortest paths. The k ASes should meet the following requirements. (i) the playback lags of the k ASes are shorter than the playback lag of the AS the newcomer located in, and (ii) the gap in the playback lag between them should be minimized.

As we know, the tracker is a very important component of a P2P system. It is responsible for maintaining a list of participating peers for each channel. In a P2P live streaming system, one tracker serves many channels, generally hundreds of them. The tracker is given another job in Agiler: find the peers from the closest upstream clusters as neighbor candidates for the newcomer. It queries the AS path server to get the closest upstream clusters. Peers are grouped into different sets according to their ASN and peers within the same AS are in one peer set. All the relative information is maintained in the tracker, as shown in TABLE I. “Peer Set” in TABLE I is a linked list consisting of peer information, including IP, port number, country name, province name, latitude and longitude. When requested by a newcomer, the tracker replies with a initial peer list including K_1 peers from the same AS and K_2 peers from the closet upstream neighbor ASes. N (on the order of 10’s, eg. 50) denotes the size of peer list, $K_1 = \alpha \times N$ and $K_2 = (1 - \alpha) \times N$. If there are not enough qualified peers, the missing peers are selected randomly.

The media server is the source peer in a P2P live streaming system, in which a live media is encoded and divided into chunks identified with chunk ID (continuously assigned sequence numbers). These chunks are injected into the system by means of being requested by other peers within the overlay.Since a media sever serves many channels, the media server can not supply one channel with unlimited bandwidth or connections. We have to impose restrictions on the bandwidth or connections that each channel consumes to ensure that every channel could obtain corresponding resource. Besides, cross-ISP traffic is limited and valuable. Fox example, in China if one peer is from ChinaCom and the other peer is from ChinaNet, the connection speed between them is very slow. Thus we could use a double-access media server, which have accesses to both ChinaCom and ChinaNet. Peers from both ISPs can connect to the media server to fetch for newly generated media chunks and this definitely could reduce the diffusion delay.

TABLE I
STRUCTURE USED IN TRACK SERVER

AS NO.	Cluster ID	Peer Number	Playback Lag	Peer Set

C. Agiler Peer Structure

The main job of an Agiler peer is to retrieve (relay) media chunks from (to) other peers and assemble them into an original streaming. Each peer cooperates with each other to keep the whole system smoothly running. Distributed cooperation contributes to the resilience and scalability of P2P applications. As depicted in Fig. 3, there are three key modules that compose an Agiler peer:

- the overlay manager
- the scheduling buffer
- the scheduler

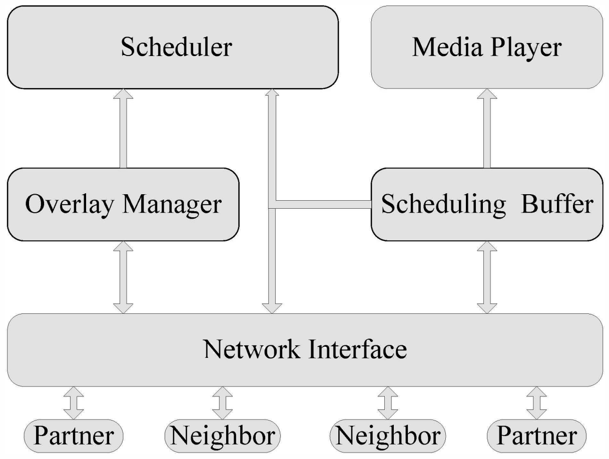

Fig. 3. Agiler Peer Structure

The overlay manager is responsible for the overlay construction. It establishes and maintains the relationships with other peers and count the traffic between itself and its neighbors. A newcomer will receive a initial peer list in the startup process. After receiving the list, the newcomer selects some peers of the list as neighbors using biased neighbor selection algorithm, which is described in section III-E.

The scheduling buffer is used to store media chunks retrieved from other peers and supply resembled chunks to the media player. Every peer in the system has a buffer organized by chunks. The availability information of chunks that is exchanged between a peer and its neighbors is called buffer map. A buffer map consists of an offset indicating the chunk ID at the buffer head and a sequence of $\{0, 1\}$ indicating the availability of chunks in the peer. For example, a value of 1 at ith position indicates that the chunk with an ID offset + $i - 1$ is stored in the buffer. We name the buffer size of a peer as the difference between the largest chunk ID advertised in the system and the smallest chunk ID advertised by the peer. All the Agiler peers have fixed buffer size and work in the fashion of sliding window.

Unlike other P2P live streaming systems, we use the partitioned buffer strategy in Agiler, in which the chunks that have just been played are reserved for sharing instead of being discarded. We name $f_p(t)$ as the ID of the chunk which is being drained by the media player of peer p at time t. As showed in Fig. 4, the scheduling buffer is partitioned into two sections by the chunk $f_p(t)$. The backward buffer stores the chunks with ID which is smaller than $f_p(t)$ and the forward buffer stores the chunks with ID larger than $f_p(t)$. Any chunk in both sections of the buffer could be requested by other peers in the system.

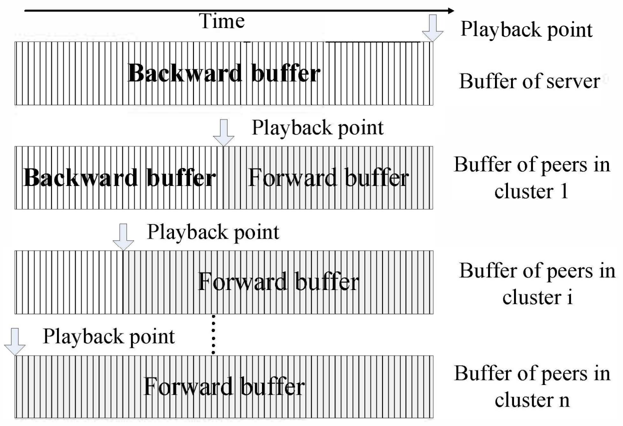

Fig. 4. Buffer Structure

The media chunks in the backward buffer could be used to serve other peers whose $f(t)$ is smaller than $f_p(t)$. For example, in Fig.2 the media chunks in the backward buffers of peers in cluster 1 could be requested by peers in cluster 2 and 3. The chunks have not been played and latest received chunks are stored in the forward buffer. In Agiler, the buffer sizes of peers within the overlay are fixed but the forward buffer size and the backward buffer size are diverse in different clusters. The peers within a cluster have the same size of both default forward buffer and backward buffer. The forward buffer size of a peer denotes the playback lag of it, which means a peer with smaller forward buffer has shorter playback lag than those who have large forward buffers. How to set the forward buffer size is described in section III-F.

The scheduler is in charge of scheduling the media chunks: requesting (sending) media chunks from (to) its neighbors. The scheduler maintains the latest buffer maps of its neighbors and gets the latest neighborhood information and statistical results from the overlay manager. Based on the above information, the scheduler determines to request which chunk from which neighbor so as to ensure the smooth playback of the peer. The scheduling strategy used in Agiler is a hybrid chunk scheduling strategy. If the first $p\%$ (default 30%) of the forward buffer is filled with chunks, rarest-first algorithm would be employed, in which the newest available chunks are to be requested. Otherwise, greedy algorithm is to be employed, in which the chunks close to the playback point are to be requested. Furthermore, the transmission time of a chunk is calculated before the chunk is requested in order to avoid unnecessary traffic. If the expected downloading finish time is later than the playback deadline of the chunk, the chunk would not be requested.

D. Locality-aware Clustering

Nowadays, the P2P traffic accounts for more than 60 percentage of total network traffic in some regions. What is more, due to the unstructured nature of P2P overlay networks, they could generate excessive network traffic, especially the unwanted cross-ISP traffic that increases ISPs' operational cost. Hence locality-awareness should be one of the essential characteristics for P2P systems.

In recent years, many locality-aware algorithms have been proposed to keep traffic locality, reduce cross-ISP traffic and improve the performance of P2P systems. Network coordinate systems (NCS) like Vilvadi [22] have been used to enhance locality-awareness in P2P systems, but it is not applicable because of its computation overhead and inaccurateness. Besides, NCS requires a substantial amount of time before it can deliver accurate information, which definitely increase the startup latency of a peer. Bindal et al. [23] proposed a new approach called biased neighbor selection to enhance BitTorrent traffic locality, in which a peer chooses its neighbors mostly from those peers within the same ISP. In [24], network views gathered at low cost from content distribution networks are used to drive biased neighbor selection without any path monitoring or probing.

Within the Internet, AS is a collection of connected Internet Protocol (IP) routing prefixes under the control of one or more network operators that present a common, clearly defined routing policy to the Internet. A unique ASN is allocated to each AS for use in BGP (Border Gateway Protocol) routing. AS numbers are important because they uniquely identifies each network on the Internet. Internet currently consists of more than $30,000$ advertised ASes [25]. It is much easier to communicate with each other for two peers in the same AS than two peers from different ASes because there is no need to change routing protocol to BGP routing and compete for the limited outbound bandwidth of ASes, especially for those ASes not operated by the same ISP. Liu et al. [26] find that locality-awareness can help existing P2P solutions to significantly decrease load on the Internet, and achieve shorter downloading time. Furthermore, they found minimizing the total number of AS hops result in significant improvement on the performance of P2P applications. In light of these facts, we take AS as the unit in our clustering strategy.

The clustering operation in Agiler processes as follows. The newcomer sends a message containing ASN, country name, longitude and latitude to the tracker. The tracker adds a corresponding item about this peer to one of the peer sets according to the ASN. The peers within the same AS form a cluster. In Fig. 5 we illustrate the locality-aware clustering using a simple network model with only 8 ASes. In this example, the media server is located in AS_1 and other peers are scattered in all the 8 ASes. Most neighbors of a peer are from the same AS and the remaining neighbors are from the closet upstream ASes. The more further the cluster is from the media server, the more larger the playback lag of the cluster is. For instance, the playback lag of peers in AS 1 is 20 seconds and the playback lag of peers in AS 2 is 25 seconds. This clustering strategy increases the matching degree of the physical network and the P2P overlay, leading to faster chunk diffusion.

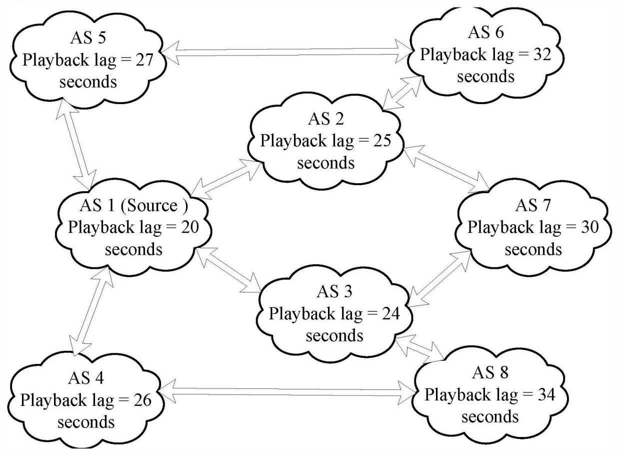
AS 5
Playback lag = 27 seconds
AS 6
Playback lag = 32 seconds
AS 2
Playback lag = 25 seconds
AS 1 (Source)
Playback lag = 20 seconds
AS 7
Playback lag = 30 seconds
AS 3
Playback lag = 24 seconds
AS 4
Playback lag = 26 seconds
AS 8
Playback lag = 34 seconds

Fig. 5. Clustering in Agiler

E. Neighbor Selection Algorithm

Biased neighbor selection is employed in Agiler to choose proper neighbors. We use both the geolocation and the round trip time (RTT) to select the closed neighbors. Htrae [27] is a good example of this kind, which combines two disparate approaches to latency prediction, network coordinate systems (NCS) and geolocation. However, Htrae is very complicated and not easy to implement, we use a simple but practical algorithm in Agiler to measure the network distance between two peers. Let $D_{i,j} = a \times G_{i,j} + b \times R_{i,j}$ be the network distance from peer i to peer j, where $G_{i,j}$ represents the geographic distance between two peers, which is calculated using the latitude, the longitude. $R_{i,j}$ represents the RTT from peer i to peer j. $G_{i,j}$ is measured in kilometers and $R_{i,j}$ is measured in milliseconds. Let Lat_i and Lon_i be the latitude and longitude of peer i measured in radian, $G_{i,j}$ could be calculated using Equation 1. There are two constants a and b where $a, b \in [0, 1]$.

$$G_{i,j} = 2\arcsin\sqrt{\sin a^2 + \cos Lat_i \times \cos Lat_j \times \sin b^2} \times 6378.1 \quad (1)$$

$$a = \frac{Lat_i - Lat_j}{2} \qquad b = \frac{Lon_i - Lon_j}{2} \quad (2)$$

An Agiler peer (we call it a host) uses Algorithm 1 to select neighbors. SN_i denotes the set of neighbors from the same AS with the host and SN_o denotes the set of neighbors from other ASes. SP_i denotes the set of partners from the same AS with the host and SP_o denotes the set of partners from other ASes. NI_{low} and NI_{high} denote the low bound and upper bound of the number of neighbors within the same AS a peer can connect. NO_{low} and NO_{high} denote the low bound and upper bound of the number of neighbors from other ASes a peer can connect. The peers who are not selected as neighbors in the overlay are called partners and they are neighbor candidates. The host keeps getting regular updates about more partners through gossip-like protocol.

Algorithm 1 Neighbor Selection Algorithm

```
Peer $p$ selects neighbors
**while** $sizeof(SN_i) > NI_{high}$ **do**
    Find the neighbor $p_i$ with the least contribution
    $SN_i \leftarrow SN_i - p_i$
**end while**
**repeat**
    Find the peer $p_i \in SP_i$ with the minimum network distance $D_{i,j}$
    $SN_i \leftarrow SN_i \cup \{p_i\}$
    $SP_i \leftarrow SP_i - p_i$
**until** $sizeof(SN_i) \geq NI_{low}$
**while** $sizeof(SN_o) > NO_{high}$ **do**
    Find the neighbor $p_i$ with the least contribution
    $SN_o \leftarrow SN_o - p_o$
**end while**
**repeat**
    Find the peer $p_o \in SP_o$ with the minimum network distance $D_{o,j}$
    $SN_o \leftarrow SN_o \cup \{p_o\}$
    $SP_o \leftarrow SP_o - p_0$
**until** $sizeof(SN_o) \geq NO_{low}$
```

In normal condition, the host calculates the network distances with its partners and sorts them in ascending order. Based on the network distance and the traffic statistical result, the host eliminate the most poorly performed neighbor and select the peers with low network distance from partners as neighbors periodically. In the startup process, only geolocation information is used when selecting neighbors because there is no time to probe the RTTs in order to reduce the startup latency. Selecting peers from the same AS as neighbors can decrease the transmission delay and keep traffic locality while selecting peers from other ASes with little gap in playback lag as neighbors can accelerate the swarm of newly generated chunks and keep the whole overlay connected tightly.

F. Proportional Playback Lag

The most significant characteristic in Agiler is that the default playback lag of a cluster is in proportion to the network distance between the cluster and the media server. Our model is motivated in part by the work of Hei et.al. [3]. In their work, the results they got demonstrate a tiering effect on the playback lag by examining buffer map traces collected from peers with different geographical locations and different network access. Another finding of them is that the playback lag of one peer is stable within a session and the lag is very large (usually more than 100 seconds).

We name the initial offset as the chunk ID that a newcomer chooses as a start point for its buffer. Initial buffering time is the interval from when the first chunk is requested until actual playback starts on the screen. The initial offset placement in P2P live streaming systems and the real deployed placement method in PPLive have been studied in [28]. The large buffer approach and the small buffer approach are the two implementations of P2P streaming systems. The large buffer approach outperforms the small buffer approach in many aspects except the large playback lag. The large buffer approach is used in Agiler skillfully with the assistance of the partitioned buffer strategy.

In a synchronized playback P2P live streaming system with a large number of peers, undesirable overlay increase the playback lag. In Agiler, peers in the same overlay are grouped into clusters according to the ASN. We adopt synchronized playback intra-cluster to accelerate the chunk diffusion and asynchronous playback among different clusters to facilitate the chunk swarming. The peers within the same AS or ASes with peering relationships with the newcomer's original AS are favored when selecting neighbors. As a result, the peers in ASes near the media server could get the media chunks they need much earlier. Thus they do not have to wait and can start to play first. How to set the default playback lag of a cluster is described in following.

The playback lag of peer i depends on the initial scheduling offset and the initial buffering time, so we can set the playback lag of a peer by tuning these two parameters. Assuming θ and τ is the initial offset and the initial buffering time respectively. Let L_p be the playback lag of peer p and $s(t)$ be ID of the newest chunk generated by the media server at time t and we have $L_p = s(t) - f_p(t)$. We use the *improved initial offset placement based on playback lag* (IPP) scheme to determine the playback lag of a cluster in Agiler. The playback lag of a cluster is calculated using Equation 3 in IPP scheme, where $f_p(t_0)$ and L_p are the playback point and playback lag of peer p respectively. Peer p is the first neighbor from upstream clusters and it reports its playback lag to peer i. α is a constant coefficient and $\alpha < 1$. We hope that that the playback lag is larger than that of peer p, so we have $f_p(t_0) + \alpha \times L_p < f_p(t_0) + r \times \tau$. The simplified form of it is $0 < \alpha < \frac{r \times \tau}{L_p}$.

$$\theta = f_p(t_0) + \alpha \times L_p \tag{3}$$

If we put Equation 3 as a mathematical form $x_{n+1} = ax_n + c$ we can obviously find that it is a contraction mapping. By doing so, the peer within clusters near the media server have short playback lag and the peers located in clusters far from the media server have long playback lag, just as illustrated in Fig. 5. The stable point of Equation 3 is $L_s = \frac{r \times \tau}{\alpha}$, so the maximum playback lag of all clusters is $L_s = \frac{r \times \tau}{\alpha}$.

As described in Algorithm 2, the playback lag of a peer is decided as follows. As soon as a peer selects a channel to watch, it sends a request to the tracker for that channel. The reply from the tracker is a message that contains a list of initial peers and the referencing playback lag. If the peer is one of the first k online peers of the cluster, the peer calculate the playback lag of the cluster using Equation 3 and feed the result back to to the tracker. The maximum one of the results is selected as the referencing playback lag of the cluster if the mean square deviation of them is less than 5 seconds. If the streaming quality of a peer is very poor, the playback lag of it would be increased by 10% each time until the streaming quality is acceptable or the playback lag is maximized.

Algorithm 2 Playback Lag Algorithm

```
$p_i$ sends a message containing AS number, longitude and latitude to the tracker $T_i$
if $p_i$ is the first $k$ online peers from cluster $C_k$ then
    Calculate the playback lag of cluster $C_k$ using Equation 3
    Record $Lag_i$ into $T_i$ database
else
    $T_i$ reads the playback lag $Lag_i$ of $C_k$ from database and sends it to $p_i$
end if
```

IV. Performance Evaluation

A. Simulation Setup

We implement an event-driven packet-level simulator coded in C++ based on the source code provided by [29] to conduct a series of simulations in this section. In this simulator, all streaming and control packets and node buffers (a node refers to a peer) are carefully simulated. For the underlying topology, we use the AS-level topology and find AS relationships collected and analyzed by CAIDA [20]. There are thousands of ASes in one file and we derive a Level-1 model of AS-level topology in China for the simulation. This model is a topology with 800 nodes distributing in 8 ASes as shown in Fig. 6. The network distance between two nodes is set to be in proportional to the distance metric of the resulting topology. $D_{i,j}$ equals to 50 if peer i and peer j are within the same AS and $D_{i,j}$ is randomly selected from $[100, 600]$ if they are from separate ASes. The details can be found in Fig. 6.

In our simulation, as commonly assumed in previous P2P system studies, we simulate an environment where the peer access links are the only bandwidth bottleneck. Therefore, in our simulation, the default streaming rate is set to 300 Kbps. Each node selects 25 other nodes as its neighbors. Each chunk has the same size of 6250 bytes. Each node estimates the bandwidth allocated from a neighbor with the traffic received from it in the past M seconds, for example, $M = 10$. The neighbor contributes the least would be replaced by a random new one with a probability of P_e (eg. 0.25). Moreover, we set the default forward buffer size to 30 seconds and the default backward buffer size to 20 seconds. The configuring parameters in detail is shown in TABLE II. In addition, we assume all peers have enough bandwidth to download the chunks for the channel they are watching. To simulate the bandwidth heterogeneity among peers, we configure the capacities of peers according to TABLE III.

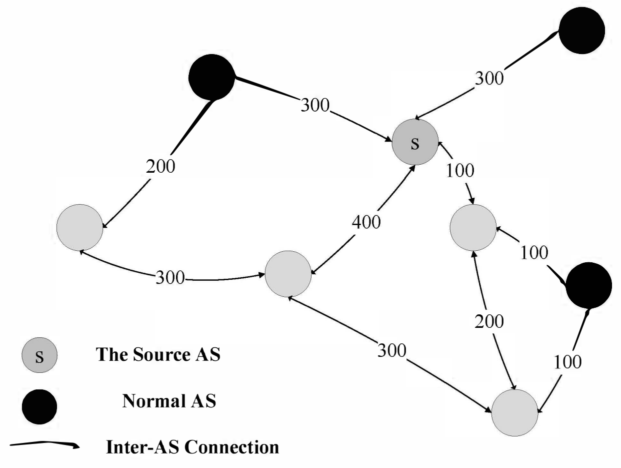

Fig. 6. Underlying Topology Used in Simulation

For comparison, we also implement a popular P2P streaming system similar to PPLive [1], one of the most popular large-scale mesh-pull P2P streaming system. Because PPLive use the proprietary protocols, so we implement this system according to the measurement studies dissecting PPLive [3, 4]. It serves as a baseline to evaluate the performance of Agiler system. In the following, the baseline system is referred as PPLive-like. In PPLive-like the large buffer approach is employed and playback lags of peers are set to be as close as possible to each other, almost to be synchronized. A peer in PPLive-like does not reserve the played chunks and thus it only has large forward buffer, the size of which denote the playback lag.

We extensively evaluate the effectiveness of the algorithms adopted in Agiler and PPLive-like through simulation and then make comparisons between them. To be fair, except the partitioned buffer strategy and the proportional playback lag strategy, other strategies adopted in Agiler and PPLive-like are the same. Random neighbor selection and synchronized playback are adopted by PPLive-like in simulation. Periodical elimination of the worst neighbor is performed in both systems. To be more precise, for each point in the figures below, we average the results by repeating 9 runs with different random seeds and data is collected when the system is stable.

B. Simulation Results

Then we show the performance of our proposed algorithms and make a comparison with the PPLive-like. First of all, let's

TABLE II
CONFIGURATION PARAMETERS

Notations	Value	Explanation
R	300 Kbps	streaming rate of a channel
NBR	25	the number of neighbors a peer has
Interval	800 ms	the interval for sending buffer map periodically
Block size	6250 bytes	the size of a media block
P_e	0.3	the probability that the neighbor contribute the least be eliminated
α	0.8	the ratio of neighbors in the same cluster to all neighbors

TABLE III
BANDWIDTH DISTRIBUTION

Type	Outbandwidth	Inbandwidth	fraction
Dsl / Ethernet	1024 Kbps	3072 Kbps	0.1
Dsl / Ethernet	512 Kbps	1024 Kbps	0.4
Dsl / Ethernet	256 Kbps	512 Kbps	0.5

introduce some simulation methodology and metrics. Static environment means that no peer quits after joining. We use Weibull(α, β) distribution with a CDF $f(x) = 1 - e^{(x/\alpha)^{\beta}}$ to randomly generate the lifetimes of the peers in dynamic environment. Given the simulation period of one run is 300 seconds, we use Weibull$(200, 2)$ distribution to generate the lifetimes of peers in the dynamic environment. And we assume the joining process of peers is a Poisson Process with rate of 20 peer per second and the maximum online user number is 800 in both environments. Moreover, to evaluate the performance, we define a metric, playback continuity. We define the playback continuity as the number of chunks that arrive at the node before playback deadline over the total number of chunks encoded in the streaming.

We first investigate the relationship between the forward buffer size and the playback lag. Fig. 7 depicts the change of the playback lag corresponding to different forward buffer sizes. We can see that the playback lag of Agiler is much less than that of PPLive-like and the difference between them get large as the forward buffer size increase. Due to the adoption of the proportional playback lag strategy and the partitioned buffer strategy, the playback lag of Agiler is in a quadratic relationship with the forward buffer size. PPLive-like adopts the large buffer approach, in which the forward buffer sizes of peers are set to be as close as possible to improve the chunk availability. The average playback lag of PPLive-like shows a linear relationship with the forward buffer size. The short average playback lag of peers in Agiler verify the effectiveness of our algorithms.

Since the average playback lag of Agiler is reduced greatly compared to PPLive-like, we may raise a question that whether the streaming quality of Agiler is affected. As shown in Fig. 8, with the growth of forward buffer size, the peers have more time to fetch the chunks to be played, so the playback continuity of both Agiler and PPLive-like get better and better in both static and dynamic environment. At the same time the playback lag grows with the forward buffer size, which is not what we want. Thus the forward buffer size should be set to a proper value where most peers can enjoy good

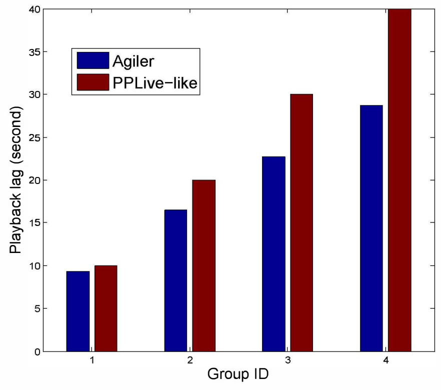

Fig. 7. Average playback continuity with respect to different forward buffer sizes, server bandwidth: 700 Kbps.

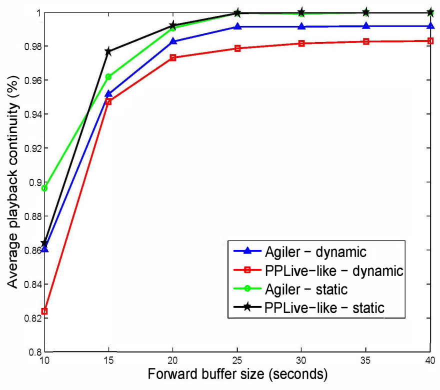

Fig. 8. Average playback lag with respect to different forward buffer sizes, server bandwidth: 700 Kbps.

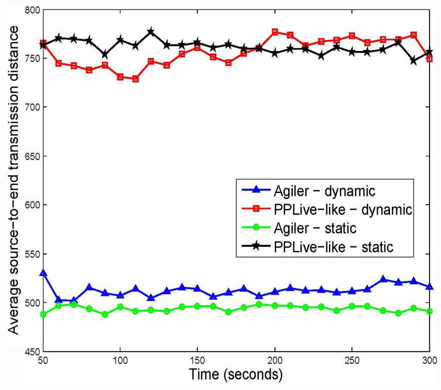

Fig. 9. Average packet distance index of media packets, server bandwidth: 800 Kbps.

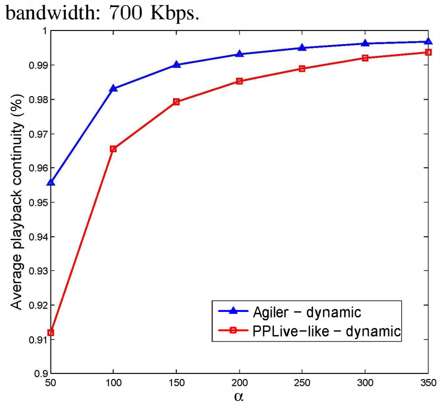

Fig. 10. Average playback continuity with respect to different α, server bandwidth: 800 Kbps.

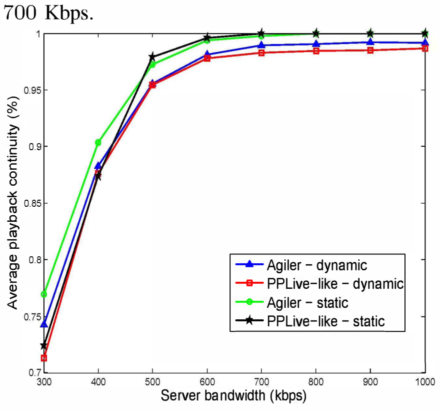

Fig. 11. Average playback continuity with respect to different server bandwidths.

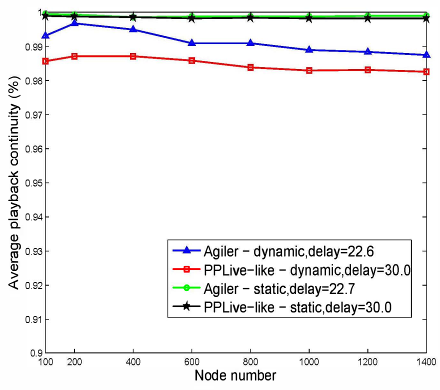

Fig. 12. Average playback continuity with respect to different node numbers, server bandwidth: 800 Kbps.

streaming quality. It is obvious that both Agiler and PPLive-like perform better in static environment than in dynamic environment. In dynamic environment, Agiler perform better than PPLive-like all the time, which indicates that Agiler can adapt to dynamic environment better. Due to the optimal overlay construction and biased neighbor selection, the impact caused by the departures of peers is reduced. So we can conclude that the reduction of the playback lag in Agiler does not affect the streaming quality.

In Fig. 9, we study the advantages of locality-awareness in Agiler compared to PPLive-like. We name *p*acket *d*istance *i*ndex (PDI) as the average network distance a media packet travels until it reaches its destination node. PDI is a significant index indicating how effective a P2P streaming system is. As shown in Fig. 9, the PDI of Agiler is much smaller than that of PPLive-like (about second three of it). The reason is that Agiler nodes are ISP-friendly and request most media chunks from nodes within the same AS while PPLive-like nodes request media chunks from random nodes in all ASes. Consequently, the time for transmitting media chunks from a node to another node is cut down and cross-ISP traffic is reduced, which lead to improvement of the system performance.

Then we study the impact of peer dynamics. We vary the churn rate by using different Weibull(α, β) distribution to generate the lifetimes of nodes. The more smaller the α is, the more higher the churn rate is and the value of α is restricted to $[50, 350]$. As shown in Fig. 10, no matter how fierce the churn rate is, Agiler performs better than PPLive-like. Even at the churn rate of Weibull$(50, 2)$, Agiler still achieves the average playback continuity of about 0.955, which indicates it is highly resistant to churn. Resilience enable the Agiler system keep smoothly working in the situation of flashing crowd that happens frequently in live streaming systems.

In Fig. 11, we explore the relationship between the playback continuity and the server bandwidth. In static environment, a server bandwidth of 700 Kbps could enable the system achieve high playback continuity up to 1 for both Agiler and PPLive-like. In the dynamic environment of Weibull$(200, 2)$, the playback continuities of both systems increase with the growth of the server bandwidth, but the rate of increment is slowing down. On the whole, Agiler outperforms PPLive-like under the same server bandwidth constraint in dynamic environment. In the situation of high peer dynamic, the increment of the server bandwidth is not a effective way to guarantee good streaming quality. So we should take the sever bandwidth allocation into consideration when deploying a P2P live streaming system.

Finally, we study the scalability of the two systems. As shown in Fig. 12, the node number is set from 100 to 1400, forward buffer size of PPLive-like is set to 30 seconds and peer lifetime distribution is generated by $\mathrm{Weibull}(200, 2)$. The performance of both Agiler and PPLive-like is very good in static environment and the increase of node number barely affects the system performance. This implies that P2P streaming systems are self-scaling and they can provide good viewing experience with limited server bandwidth. In dynamic environment, Agiler outperforms PPLive-like with a large gap in both the playback continuity and the playback lag. When the node number is small (eg. 100), the increase of node number is helpful to both systems. However, when the node number is up to 1000, the system performance is deteriorated slightly because large number of nodes require more server bandwidth to recover from peer dynamics. The locality-aware overlay construction and the proportional playback lag strategy contribute to the scalability of Agiler.

V. Conclusion

In this paper we present the design and performance evaluation of a novel P2P live streaming system Agiler, with objective of reducing the playback lag under a tight severe bandwidth constraint. Peers are grouped into clusters according to the ASN and playback lags of the clusters are set according to the network distance between the clusters and the media server. Besides, biased neighbor selection and asynchronous playback among clusters are employed in Agiler, which could improve the network efficiency and lead to a better viewing experience. Simulation results show that Agiler reduce the average playback lag by as much as more than 20% compared with traditional systems.

Future work might be extended in several directions. First, more efforts should be spent on how to determine the playback lag of a cluster so as to achieve good viewing experience without jitters. Second, peers with priority should be taken into consideration separately. We could reduce the playback lag of peers with priority by assigning it to a cluster near the media server regardless of the geographical location. Third, how to efficiently allocate the server bandwidth under multi-channel scenarios is a quite challenging problem. More server bandwidth should be allocated to lag-sensitive channels to shorten the playback lag and less bandwidth is allocated to lag-insensitive channels just to ensure the smooth playback.

VI. Acknowledgements

This work was partially supported by NSFC under grant60803119, by Science and Technology Commission of Shanghai Municipality under grant 08dz150010E, and by Shanghai Educational Development Foundation under grant 2007CG07. Jin Zhao is the corresponding author.

References

[1] "PPLive," http://www.pplive.com/.

[2] "PPStream," http://www.ppstream.com/.

[3] X. Hei, Y. Liu, and K. W. Ross, "Inferring Network-Wide Quality in P2P Live Streaming Systems," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 9, pp. 1640–1654, 2007.

[4] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt, "Mapping the pplive network: Studying the impacts of media streaming on p2p overlays," UIUC, Tech. Rep., August 2006.

[5] S. Ali, A. Mathur, and H. Zhang, "Measurement of commercial peer-to-peer live video streaming," in *In Proc. of ICST Workshop on Recent Advances in Peer-to-Peer Streaming*, Weaterloo, Canadda, 2006.

[6] X. Zhang, J. Liu, B. Li, and T. P. Yum, "CoolStreaming/DONet: A Data-driven Overlay Network for Efficient Live Media Streaming," in *Proc. of IEEE INFOCOM 2005*, 2005.

[7] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang, "A Peer-to-Peer Network for Live Media Streaming: Using a Push-Pull Approach," in *Proc. Of ACM Multimedia 2005*, Nov. 2005.

[8] F. Pianese, D. Perino, J. Keller, and E. Biersack, "PULSE: An Adaptive, Incentive-Based, Unstructured P2P Live Streaming System," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1645–1660, Dec. 2007.

[9] M. Wang and B. Li, "R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1655–1666, Dec. 2007.

[10] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in cooperative environments," in *Proc. of SOSP'03*, 2003.

[11] Z. Liu, Y. Shen, S. S. Panwar, K. W. Ross, and Y. Wang, "Using layered video to provide incentives in p2p live streaming," in *Proceedings of P2P-TV*. New York, NY, USA: ACM, 2007, pp. 311–316.

[12] D. Ren, Y.-T. H. Li, and S.-H. G. Chan, "On Reducing Mesh Delay for Peer-to-Peer Live Streaming," in *Proc. of IEEE INFOCOM 2008*, 2008.

[13] Y. Liu, "On the Minimum Delay Peer-to-Peer Video Streaming: how Realtime can it be?" in *Proc. of ACM Multimedia*, 2007.

[14] C. Feng, B. Li, and B. Li, "Understanding the performance gap between pull-based mesh streaming protocols and fundamental limits," in *Proc. INFOCOM 2009. The 28th Conference on Computer Communications. IEEE*, 19–25 April 2009, pp. 891–899.

[15] M. Zhang, L. Sun, , and S. Yang, "iGridMedia: Providing Delay-Guaranteed Peer-to-Peer Live Streaming Service on Internet," in *Proc. of IEEE Globecom 2008*, 2008.

[16] D. Wu, C. Liangz, Y. Liuz, and K. Rossy, "View-Upload Decoupling: A Redesign of Multi-Channel P2P Video Systems," in *Proc. of IEEE INFOCOM 2009*, 2009.

[17] X. Zhang, X. Chen, N. Ren, J. Zhao, and X. Wang, "Sonicstream: An implementation of a live p2p media streaming system with improved playback lag," in *Proc of IEEE ICCE 2009.*, 2009.

[18] D. Huang, J. Zhao, X. Wang., "Trading bandwidth for playback lag: Can active peers help?" in *Proc. of ACM Multimedia, Short paper*, 2010.

[19] "pwhois," http://pwhois.org/.

[20] CAIDA Project. The CAIDA AS Relationships Dataset. http://www.caida.org/data/active/as-relationships/.

[21] D. Eppstein and S. J. C. C, "Finding the k shortest paths," *SIAM Journal on Computing*, vol. 28, pp. 652–673, 1998.

[22] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *In SIGCOMM*, 2004, pp. 15–26.

[23] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in bittorrent via biased neighbor selection," in *Proceedings of ICDCS*, 2006.

[24] D. R. Choffnes and F. E. Bustamante, "Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems," in *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4. New York, NY, USA: ACM, 2008, pp. 363–374.

[25] G. Huston. The 32-bit as number report, december 2009. http://www.potaroo.net/tools/asn32/.

[26] B. Liu, Y. Cui, Y. Lu, and Y. Xue, "Locality-awareness in bittorrent-like p2p applications," *IEEE Transactions on Multimedia*, vol. 11, no. 3, pp. 361–371, April 2009.

[27] S. Agarwal and J. R. Lorch, "Matchmaking for online games and other latency-sensitive p2p systems," in *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. New York, NY, USA: ACM, 2009, pp. 315–326.

[28] C. Li and C. Chen, "Initial offset placement in p2p live streaming systems," *CoRR*, vol. abs/0810.2063, 2008.

[29] M. Zhang, "Peer-to-Peer Streaming Simulator," http://media.cs.tsinghua.edu.cn/ zhangm/download/, 2007.