A Measurement of a Large-scale Peer-to-Peer Live Video Streaming System

Susu Xie, Gabriel Y. Keung, and Bo Li

Department of Computer Science
Hong Kong University of Science and Technology, Hong Kong, China

***Abstract*— Peer-to-Peer (P2P) technologies have found much success in applications like file distributions, and its adoption in live video streaming has recently attracted significant attentions. With the emerge of commercial P2P streaming systems that are orders of magnitude larger than the academic systems, understanding its basic principles and limitations are important in the design of future systems.**

Coolstreaming represented one of the earliest large-scale live streaming trials in the Internet. In this paper, we discuss the fundamental design principles and examine the system dynamics. By leveraging the recent results obtained from live event broadcast, we develop some basis to demonstrate that a random partnership selection has the potential to scale. Specifically, first, we examine the overlay topology and how it converges. Second, using a combination of real traces and analysis, we quantitatively provide insights on how buffering technique resolves the problems associated with dynamics and heterogeneity. Third, we discuss the main limitations and the implications on the scalability.

Key Words: Video streaming, peer-to-peer technology, measurement, video over IP

Technical Area: Peer-to-Peer Technologies

I. Introduction

Recently there has been significant interest in adopting the Peer-to-Peer (P2P) technology for Internet live video streaming [1]-[2]. There are primarily two factors fueling this development: First, a P2P system requires minimum support from the Internet, thus it is cost-effective and easy to deploy. Second, in such a system, each peer node that participates in a video program is not only downloading content, but also uploading to other participants watching the same program. Consequently, it has the potential to scale as greater demand also generates more resources.

Coolstreaming represented one of the earliest large-scale peer-to-peer video streaming experiments [3], which was built on the notion of *data-driven*, somewhat similar to the technique used in BitTorrent but with much more stringent timing and rate constraints. The key idea is that every node periodically exchanges its data availability information with a set of partners, and retrieves unavailable data from one another. The system demonstrated excellent self-scaling property over the global Internet, in which the earlier experiment reported a peak of 80,000 concurrent users with streaming rates over 400 Kbits/sec. Since then, systems such as PPlive [4], Sopcast [5] and PPstream [6] have also been developed and deployed. However, question remains whether a P2P streaming system can really scale to millions of viewers. Perhaps more importantly, the question is what set of challenges and possible technical solutions are for these system to scale with the QoS constraints. We believe understanding the fundamental design principles and system dynamics is an important step toward their further development. Moreover, there has been few experimental studies on large-scale P2P streaming systems. The main constrain in prior studies was the lack of internal knowledge in the architecture and control mechanisms, due to the proprietary nature of commercial systems, in which few measurement studies [7]-[8] could only seek for mechanisms at network edges, such as packet sniffing, to capture the external behaviors of such systems. This, however, often fails to provide insights into the fundamental design trade-offs and to offer rational explanations for the engineering decisions.

Our study in this paper differs from all prior works in that it examines the workload, performances and system dynamics based on an internal logging system we have designed, with full knowledge of the architecture and control mechanisms. Specifically, we leverage a large set of traces we obtained from live streaming events using Coolstreaming on 27^{th} September, 2006. The main contributions from this study are: 1) we describe the fundamental design principles and trade-offs in Coolstreaming system; 2) based on a simple topology model, we illustrate how a random peer selection algorithm can lead to the topology convergence; 3) using the set of real traces and analysis, we quantitatively provide the insights on how buffering technique resolve the problems associated with dynamics and heterogeneity; 4) we discuss the fundamental limitations in such a system and the implications on the scalability.

The rest of this paper is organized as follows. Section II briefly reviews the related works. Section III describes the basic architecture of the Coolstreaming system and the key components. Section IV discusses the system dynamics, in particular peer joining and peer adaptation processes. Section V presents results from live event broadcast and examines the performance implications. Section VI concludes the paper with highlights on future research.

*The research was support in part by grants from RGC under the contracts 616505, 616406 and 616207, by a grant from HKUST under the contract RPC06/07.EG27, by grants from NSF China under the contracts 60429202 and 60573115, by a grant from National Basic Research Program of China (973 Program) under the contract 2006CB303000

II. Related Works

Earlier Internet video streaming systems were largely built on the *IP multicast* model [9]. This model, while being efficient in data delivering, encountered a number of practical problems. Perhaps most significantly it requires each router to maintain state, which violates the "state-less" architecture principle and also brings difficulty in the design of such high level functionalities as error, flow, and congestion control. Later, researchers advocated moving multicast functionalities away from routers towards end systems [10]-[12], in which multicast functions like group membership and multicast routing were implemented at end systems assuming only unicast service underneath. A node participate in multicast via an *overlay structure*, in which each of its edges corresponds to a unicast path with another such node. It has been demonstrated in small scale that it is feasible to implement such multicast functions at end systems while keeping the core functionalities of the Internet intact.

We make a distinction in this paper by referring Coolstreaming as a data-driven P2P streaming system, in which there is no explicit overlay topology construction. We refer to the other approaches as *tree-based overlay multicast*, given the explicit construction and maintenance of multicast tree(s). This can be in the form of a single tree [11][12] or multi-trees [13][14]. Data-driven systems do not explicitly construct and maintain an overlay structure and often adopts a gossip-like protocol to locate a peer with video content. Such a system has demonstrated great potential to scale in the global Internet, as proven by Coolstreaming [3], PPlive [4], Sopcast [5] and PPstream [6], which have attracted millions of viewers.

The success of Coolstreaming and similar systems has attracted significant research interests. For example, Francis et al. proposed an architecture called Chunkyspread [15] , in which a randomized multi-tree is constructed to spread the slices of the video stream and that the topology could be improved iteratively by swapping parents with respect to the load and delay measurement. In [16], Reza et al. examined the data-driven P2P streaming system in a static setting and showed that swarming and diffusion can be efficient for content delivery. Recently, many measurement works have also been done to understand these complex systems. In [7], the performance of PPLive was measured using passive packet sniffing and presented several interesting insights on the streaming performance and workload characteristics. Another measurement was done in [8] for PPLive and SopCast, which provides additional observations on the stability of the system and the cost of the download. Other recent measurement studies can be found in [7][17]-[19]. However, there is a major constraint in these studies, that is, the lacks internal knowledge of system architecture. Hence, such measurements could not completely reveal the internal dynamics and basic design trade-offs.

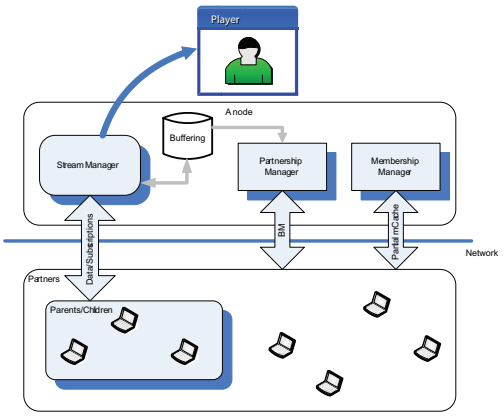

Fig. 1. Coolstreaming system diagram

III. Basic Architecture

Coolstreaming was developed in Python language earlier 2004. This implementation is platform independent and its supports RealPlayer and Window media formats. Since the first release (Coolstreaming v0.9) in March 2004, it has attracted millions of downloads worldwide. The peak concurrent users reached over 80,000 with an average bit rate of 400Kbps. From our previous work, [20] summarized the distributions of the IP addresses obtained from worldwide.

A. Basic Components

Fig. 1 depicts the current system design for Coolstreaming. There are three basic modules in the system: 1) *Membership manager*, which maintains partial view of the overlay. 2) *Partnership manager*, which establishes and maintains TCP connections, or partnership, with other peer nodes. It also exchanges the availability of stream data in the *buffer map* (BM) with the peer nodes, which we will explain later. 3) *Stream manager*, which is the key component for data delivery. Besides providing stream data to the media player, it also makes decisions on where and how to retrieve stream data. The central design in this system is based on the *data-driven* notion, in which every peer node periodically exchanges its data availability information with a set of partners to retrieve unavailable data, while also supplying available data to others. The fundamental advantage of such an approach is the elimination of explicitly constructing and maintaining any specific overlay network. This also offers a few other advantages: 1) *easy to deploy*, as there is no need to maintain any global structure; 2) *efficient*, in that data forwarding is not restricted by the overlay topology but by its availability; 3) *robust and resilient*, as both the peer partnership and data availability are dynamically and periodically updated.

B. Overlay Construction and Maintenance

Each node in the system has an unique identifier and maintains a membership cache (*mCache*) containing a partial list of the currently active nodes in the system. The gossip protocol is used for overlay construction. Such a technique has been widely used in BitTorrent and other

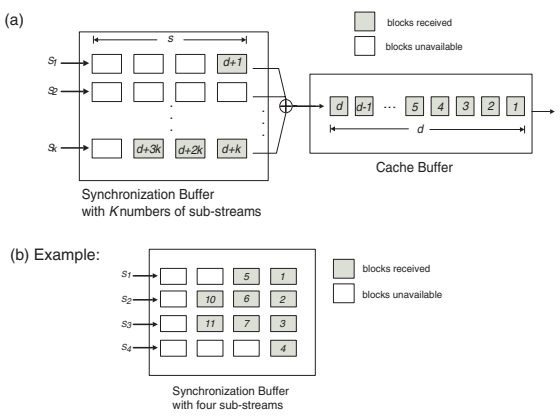

Fig. 2. (a) Structure of buffers in a node, (b) An example of combination process in Synchronization buffer

P2P system [21]-[22]. The random choice of the gossip algorithm achieves excellent resilience against random failures and also enables decentralized operation. Specifically, in Coolstreaming, a newly joined node contacts a boot-strap node for a list of peer nodes and stores that in its own mCache. It then randomly selects a few nodes from this list to establish *partnership* maintained by the partnership manager module in Fig. 1. In other words, the partnership specifies that two peers can exchange video availability information with each other. A *parent-children relationship* can be established when a node (the child) is actually receiving video content from another node (the parent). Apparently, the parent-children nodes are a subset of the nodes from the partnership.

Partnership can be broken from time to time due to many reasons and nodes will need to perform *partner re-selection* to maintain the continuity of video streams. For example this occurs when a node is receiving insufficient bit rates from its partners or the departure of a parent node (churn), in which cases the node has to drop some partners and re-establish partnership with other peers.

C. Sub-streams and Buffering

The video stream is divided into multiple *sub-streams*, and nodes could subscribe to sub-streams from different partners. Each sub-stream is further divided into *blocks* with equal size, in which each block is assigned a sequence number representing its playback order. Each node maintains an internal buffer, whose structure is illustrated in Fig. 2a. It is composed of two parts: *synchronization buffer* and *cache buffer*. A received block is firstly put into the synchronization buffer for each corresponding sub-stream. They will be combined into one stream when blocks with continuous sequence numbers have been received from each sub-stream. An example is given in Fig. 2b, in which the combination process stops at the third sub-stream while it awaits the block from the 4^{th} sub-stream with sequence number 8.

A *buffer Map* or BM is introduced to represent the availability of the latest blocks of different sub-streams in the buffer. This information also has to be exchanged periodically among partners in order to determine which sub-stream(s) to subscribe to. Specifically, BM is represented by a $2K$-tuple, where K is the number of sub-streams. The first K components of the tuple records the sequence number of the latest received block from each sub-stream. The second K components of the tuple represents the subscription of sub-streams from the partner.

R	bit rate of the live video stream
K	number of sub-streams
B	length of a peer's buffer in unit of time
T_s	out-of-synchronization threshold, i.e. upper bound of acceptable deviation between sub-streams
T_p	maximum allowable latency for a partner behind others
T_a	the period a peer re-select a parent if needed
D_p	the out-going sub-stream degree of node p

TABLE I
SYSTEM PARAMETERS OF COOLSTREAMING

IV. Inside Look into System Dynamics

In this section, we discuss the system dynamics including peer joining, peer adaptation and relevant timing. Table I summarizes the parameters and notations.

A. Peer Joining

A newly joined node first contacts the boot-strap node for an initial list of nodes that it expects to establish partnership and stores the information in its mCache. With the exchange of BM information, the newly joined node can obtain the video availability information from a set of randomly selected nodes from the list. As we have discussed before, the node then needs to determine the initial sequence number of the block that it will start to retrieve.

Suppose the range of the blocks available (i.e., the sequence number of the video blocks) in all its partners are from n to m; intuitively, the node should request from a block with sequence number somewhere in the middle. The rationale for this is if the node requests the block starting from the largest sequence number m, the partner nodes might not have sufficient follow-up blocks to satisfy the continuity requirement for the video stream; on the other hand, if the node requests the block from the lowest sequence number n, this can result in two problems: 1) such blocks might no longer be available once it is pushed out of the partners' buffer due to the playout; 2) it might take considerable amount of time for the newly joined node to catch up with the current video stream, which would incur long initial delay.

From the above arguments, in the Coolstreaming system, a node subscribes (i.e., pulls) from a block that is shifted by a parameter T_p (to be defined in next subsection) from the latest block m. Once the initial sequence number is determined, the node checks the availability of blocks in its partners' BM and then selects appropriate partner nodes as its parents for each sub-stream.

B. Peer Adaptation

Since congestion and peer churns occur frequently in the Internet, it is important for any P2P system to do peer adaptation, i.e., to search for new parent(s).

In simple terms, a peer node needs to constantly monitor the status of the on-going sub-stream transmissions and re-selects new parents when existing TCP connections are inadequate in satisfying the streaming requirement. The questions are what triggers this adaptation, i.e., how to detect possible congestion or churns, and how to re-select new parent(s).

The status of insufficient upload capacity could be detected by the following metric: First, we introduce two thresholds $\{T_s, T_p\}$, which are related to the sequence number of blocks for different sub-streams in each node (say, node A). T_s can be interpreted as the threshold of the maximum sequence number deviation allowed between the latest received blocks in any two sub-streams in node A. T_p is defined as the threshold of the maximum sequence number deviation of the latest received blocks between the partners and the parents of node A. We denote $H_{S_i,A}$ as the sequence number of the latest received block for sub-stream S_i at node A. For monitoring the service of sub-stream j by corresponding parent p, two inequalities can be introduced

$$\max\{|H_{S_i,A} - H_{S_j,p}| : i \leq K\} < T_s \quad (1)$$

$$\max\{H_{S_i,q} : i \leq K, q \in partners\} - H_{S_j,p} < T_p \quad (2)$$

Inequality (1) is used to monitor the buffer status of received sub-streams for node A. If this inequality does not hold, it implies that at least one sub-stream is delayed beyond threshold value T_s. This is an indication for either congestion or insufficient upload capacity for this sub-stream, which will subsequently trigger peer adaptation. The second Inequality (2) is used to monitor the buffer status in the parents of node A. Node A compares the buffer status of current parents to that of its partners. If this inequality does not hold, it implies that the parent node is considerably lagging behind in the number of blocks received when comparing to at least one of the partners, which currently is not a parent node for the given node A. This will also trigger node A to perform peer adaptation by switching to a new parent node from its partners.

When a peer selects a new parent from its partners either due to peer adaptation or new joining, the selected partner must satisfy the two inequalities. If there is more than one qualified partners, the peer will choose one of them randomly. A parent node however will always accept requests and it will simply push out all blocks of a sub-stream in need to the requesting node. Apparently such a peer adaptation can potentially cause stream disruption and instability of the overlay topology. A cool-down timer is introduced to confine nodes to perform peer adaptation once only within a cool-down period of time T_a.

Since the selection of a new parent is based on Inequalities (1) and (2), here we only consider the sequence numbers instead of the parent's available upload bandwidth. In that case, a parent with insufficient upload bandwidth can be selected, and its upload capacity can be insufficient to support all children nodes. This is because the parent will continue accepting new children as long as its total number of partners is less than the upper bound M introduced earlier. In this case, all children nodes have to compete for the insufficiently aggregated upload capacity from the parent. We call this situation *peer competition*, in which eventually one or more nodes will lose and trigger peer adaptation. This causes a chain reaction of peer adaptations. During the process, the inequalities can be violated and some temporary parents may be selected and abandoned before a capable parent is located.

C. System Dynamics

In this subsection, we discuss the relative timing involved in the peer adaptation process. The average bit rate required for one single sub-stream transmission is R/K. Consider the case that node p (the parent) is pushing data to node q (the child). Suppose that node p is capable of providing upload bandwidth (i.e., bit rate) $r_\uparrow > R/K$, in this case node q can eventually catch up with node p in terms of the missing blocks from a sub-stream. This can be referred to as a *catch up process*. Let us assume initially there are l blocks missing from node q comparing to node p, then time $t_\uparrow$ for the catch up process can be easily computed by

$$\begin{aligned} r_\uparrow \cdot t_\uparrow &= R/K \cdot t_\uparrow + l \\ t_\uparrow &= \frac{l}{r_\uparrow - R/K} \end{aligned} \quad (3)$$

Now, let us examine the chain reaction of peer adaptation. Suppose the parent of node p is incapable of supporting its children nodes including p; p loses the competition and if p cannot find a new capable parent fast enough, its children (such as node q) can be stalled. The time, is denoted as $t_\downarrow$, for a child q of parent p to abandon p as its parent, i.e., the sub-stream obtained from node p in node q lags behind other sub-streams beyond threshold T_s. In other words, if p can find a new capable parent within time $t_\downarrow$, there is no need for node q to perform peer adaptation.

A node can still receive blocks from those temporary parents, suppose with an average bit rate $r_\downarrow$, and $r_\downarrow < R/K$. Then time $t_\downarrow$ can be computed by

$$\begin{aligned} t_\downarrow \cdot r_\downarrow + l &= t_\downarrow \cdot R/K \\ t_\downarrow &= \frac{l}{R/K - r_\downarrow} \end{aligned} \quad (4)$$

Suppose the sub-stream degree of p is D_p when q is accepted as a child, and p can satisfy all its children before q's subscription. After q is accepted, the upload bandwidth for each sub-stream transmission of p decreases from R/K down to $r_\downarrow$, where

$$r_\downarrow = \frac{D_p}{D_p + 1} \cdot R/K \quad (5)$$

The result of competition depends on the buffer status of the children nodes at the beginning of the competition.

Suppose it takes t_{lose} time for one of the children to lose the competition due to a subscribed sub-stream lagging behind others from t_δ to T_s in unit of blocks, i.e., it violates Inequality (1). We have

$$\begin{aligned}(T_s - t_\delta) &= R/K \cdot t_{lose} - \frac{D_p}{D_p+1} \cdot R/K \cdot t_{lose} \\ t_{lose} &= \frac{(D_p+1)(T_s - t_\delta)}{R/K}\end{aligned}$$

As discussed earlier, the system confines nodes to perform peer adaptation once only in time period T_a. Node q will unsubscribe from p if there is no other children of p losing the competition within T_a. Thus we can calculate the probability for a child to lose the competition by

$$\begin{aligned}P(t_{lose} \leq T_a) &= P(\frac{(D_p+1)\cdot(T_s - t_\delta)}{R/K} \leq T_a) \\ &= P(t_\delta \geq T_s - \frac{T_a \cdot R/K}{D_p+1}) \qquad (6)\end{aligned}$$

V. Results

In this Section, we present the results obtained from live streaming events using Coolstreaming on 27^{th} September, 2006. We will describe the internal logging system and data collection method followed by results and discussions on a wide range of system performances. Specifically, we will examine the overlay topology, user distributions, video playback quality and sensitivity against system size, user types and system dynamics. Finally we discuss the issues related to scalability.

A. Log and Data Collection

In this subsection, we describe the system configuration, log system and format. Each video program is streamed at a bit rate of 768 Kbps, a typical rate for TV-quality streaming. The users contact a web server to select the program that they intend to watch. We use an ActiveX component in JavaScript code to collect the peer activities as well as status information and reports back to a log server. To provide better streaming service, we have also deployed a set of dedicated servers (24) with 100 Mbps connections. The source sends video streams to the servers, which are collectively responsible for streaming the video to peers. From the system's point of view, this improves the overall performance in two ways: 1) the streaming capacity is steadily amplified with the deployment of the servers; 2) the servers can be placed strategically, thus the content is closer to the users.

We placed a dedicated log server in the system. Each user reports its activities to the log server including events and internal status periodically. The users and the log server communicate with each other using the HTTP protocol. The log server stores the reports received from peers into a log file. Each log entry in the log file is a normal HTTP request URL string referred as a *log string*. The information from a peer is compacted into several parameter parts of the URL string and reported to the log server. The URL string contains various number of data blocks, which are formed in "name=value" pairs and separated by "&". Reports from peers can be divided into two classes. The first class is *activity report*, which indicates the peer activities such as join and leave. Activity reports are sent out immediately when the corresponding activities take place. The second class is *status report*, which indicates the internal state of peers sent out every 5 minutes periodically. There are three types of status reports:

- A *QoS report* records the perceived quality of service, for example, the percentage of video data missing at the playback deadline;
- A *traffic report* records the amount of video content downloaded and uploaded;
- A *partner report* records partner activities. Since the nodes might change partners frequently, we use a compact report that records a series of activities to reduce log server's load.

B. Conceptual Overlay

There is a need to classify user connection types in order to determine the *uploading capacity distribution*. This is primarily based on the local information such as the IP address and partnership status, thus errors can occur. Based on their IP addresses, we can classify the users into private or public users. By checking whether they are successful in establishing TCP connections or not, we can further classify users into the following four types:

- *Direct-connect*: peers have public addresses with both incoming and outgoing partners;
- *UPnP*: peers have private addresses with both incoming and outgoing partners. In real systems, peers can be aware of UPnP devices in the network since they will explicitly acquire public IP addresses from the devices;
- *NAT*: peers have private addresses with only outgoing partners;
- *Firewall*: peers have public addresses with only outgoing partners.

Consider a peer A with its partner B. Peer B is an *incoming partner* if B initiatively establishes partnership with A. On the other hand, B is an *outgoing partner* if the partnership establishment of A is initialized by itself. If peer A with a private IP address has only outgoing partner for a relatively long period of time, it can be categorized as NAT user. However, once a NAT or firewall user established a partnership with another node, it can still upload video to other node whenever it is capable of doing so. In other words, a NAT or firewall user can become the parent for another node.

We plot the users according to their categories and the upload contributions in Fig. 3. Observed from the figure we can see uneven contributions from peer nodes in the P2P streaming system. Specifically, 30% or so peer nodes in the overlay, i.e., nodes under UPnP and direct-connect,

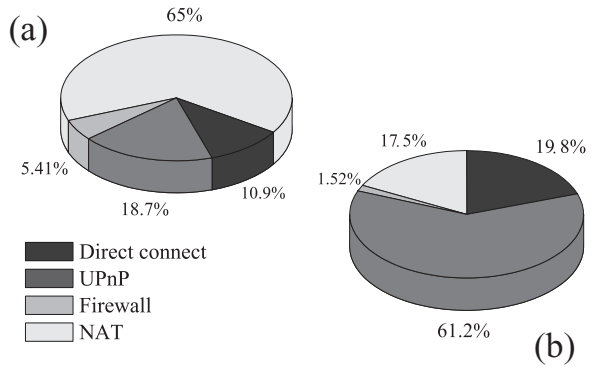

Fig. 3. (a) User type distribution; (b) User upload bytes contribution distribution.

contribute more than 80% of the upload bandwidth. This is consistent from the results obtained in [23].

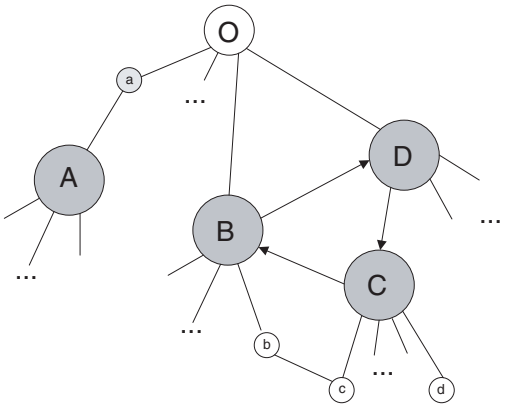

Fig. 4. Conceptual view of live video streaming overlay

It is usually difficult to capture the exact snapshot of the overlay topology in a real system. Based on the above observations, we conjecture a conceptual overlay in Fig. 4. Node O is the source node, peers $\{A, B, C, D\}$ are direct-connect/UPnP nodes and others are nodes behind NAT and firewall. The overlay is illustrated as a mesh structure, but generally resembles a tree-like topology with a few random links between selected peers (this will be further explained). Each link represents one or more sub-stream transmission. We next discuss peer classification and the basic property of the overlay topology.

1) Peer Classification: Some of direct-connect/UPnP peers often stay in the overlay for long time and have relatively large amount of upload bandwidth. Children of a direct-connect/UPnP peers can obtain sufficient upload bandwidth for sub-stream transmissions, thus can be fairly stable. As we will describe later, there are usually less chances for peer competition among the children belonging to a direct-connect/UPnP peers. The degree of a direct-connect/UPnP peers often reaches the maximum allowed by the system. Peers $\{A, B, C, D\}$ are such direct-connect/UPnP peers in Fig. 4.

Some of NAT or firewall users also tend to stay in the overlay for long time but have only limited upload bandwidth. The contribution to the overlay is restricted and its children suffer from peer competitions. However, they could stay close to the source. Peer $\{a\}$ is one of such peers.

The remaining peers are those that usually stay in the system for a short period of time, often with very little upload capacity. Most of them are NAT/firewall peers. They are often positioned far from the source and have to re-select parent relatively often. Peers $\{b, c, d\}$ are such peers in the figure.

2) Overlay Structure: There are several interesting properties in the conceptual view of the overlay in Fig. 4.

- Large amount of peers tends to clog under direct-connect/UPnP peers;
- Connections among NAT/Firewall peers (we refer as random links), i.e. b and c in the figure, are relatively rare.

We next explain the rationale behind these properties. The overlay of the streaming system is subject to change, i.e., peer adaptations occur from time to time. A peer node tends to be stable under two conditions. First, the parent node can provide sufficient upload bandwidth and it itself is stable. Second, according to Eq. (6), the larger sub-stream degree of the parent, the less probability that the children will lose when competition happens; in another word, if a node subscribes to a NAT/Firewall parent, whose sub-stream degree is often relatively small, the node tends to be more vulnerable. Usually even if a peer selects a NAT/Firewall peers as the parent at the beginning, as it suffers from insufficient upload bandwidth and is frequently subject to peer adaptation, eventually it can convert to a direct-connect/UPnP peers for its parent.

The stability of a direct-connect/UPnP peers also explains why peers tend to clog under direct-connect/UPnP peers. For example, if a peer select a NAT/Firewall peers as the parent, like $\{A, a\}$ in Fig. 4, such subscription can be easily broken due to the insufficient upload bandwidth from node a. The peer will eventually be stabilized under some direct-connect/UPnP peers. If the system runs long enough, most of peers will likely become children of direct-connect/UPnP peers.

A natural conclusion can be drawn from this overlay structure is that the video quality perceived by nodes heavily depends on that perceived by direct-connect/UPnP peers. The existence of direct-connect/UPnP peers in the P2P streaming system is not only critical for overlay stability, but also helps to reduce the latency. As discussed earlier (see Eq. (3)), the catch-up process can be significantly speeded up with the selection of a direct-connect/UPnP peers as a parent node. Another implication is that the source needs to be placed in a node with sufficient upload capacity; otherwise frequent peer adaptations can occur that make the overall system unstable.

C. Session Level Performance

The *session* captures a user activity when a user joins the system until it leaves the system. In each session, a

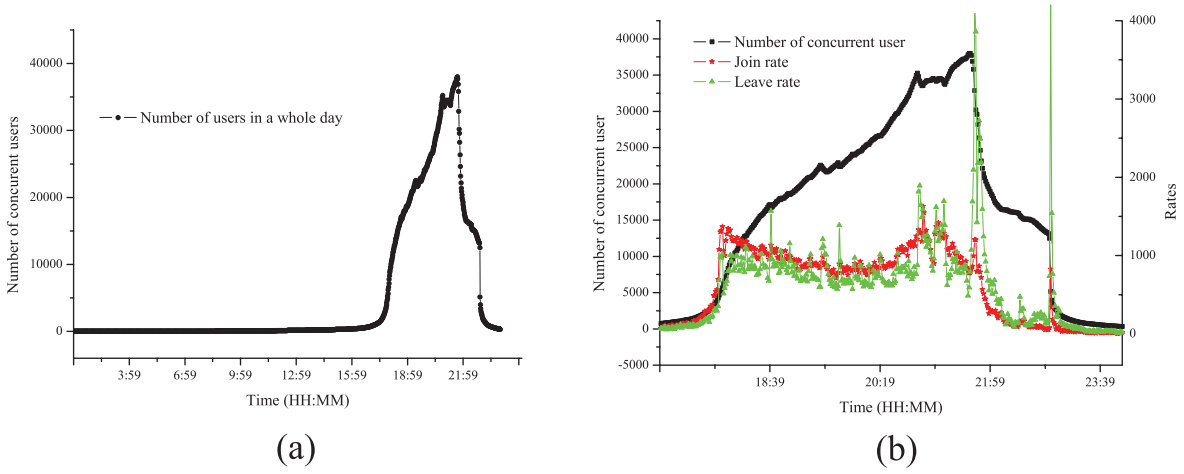

Fig. 5. (a) The evolution of the number of users in the system in a whole day; (b) The evolution of the number of users in the system from 18:00 to 23:59

client (user) reports up to four events to the Log server:

- *Join* event: This event is reported when the client joins the system and connects to the Booting Server.
- *Start subscription* event: This event is reported as soon as the client establishes partnership relations with other clients and starts receiving (video) data from its parent node(s).
- *Media player ready* event: This event is reported when the client receives sufficient data to start playing.
- *Leave* event: This event is reported when the user leaves the system.

For each pair of join/leave event, a *session* is counted. The *session duration* is the time between join and leave events. For a *normal session*, the sequences of reported events include: (1) join event, (2) start subscription event, (3) media player ready event, and (4) leave event, as defined earlier. The *media player ready time* is the time between join and media player ready events; the *start subscription time* is the time between join and start subscription events.

Fig. 5a shows the number of sessions (concurrent users) in the system in a particular day, and Fig. 5b plot the number of sessions in the system between 18:00-24:00. These illustrate user behavior in a typical weekday and at the peak hours in the evening. The sudden drop in the number of users around 22:00 is caused by the ending of some programs. This also shows that the system can scale well with the current setting as the number of nodes in the system can quickly ramp up to $40,000$.

Fig. 6 plots the distribution of the start subscription time and media player ready time. There are two observations from this result: (1) many users could successfully find a capable parent and receive sufficient video content to start playing the video within a short period of time; (2) on the other hand, this exhibits heavy-tailed distribution, which indicates that there are also a large number of users that fail to find a capable parent, and thus can

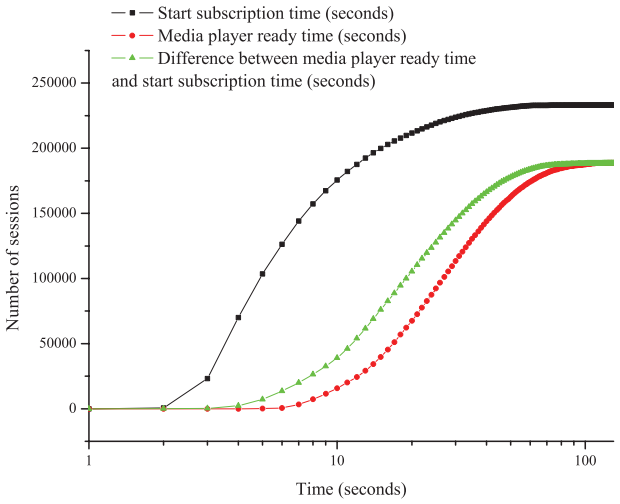

Fig. 6. The comparison between start subscription time, media player ready time and their difference

not obtain the video program in time. Fig. 6 also plots the cumulative distribution of the difference between the media player ready event and the start subscription event, which essentially indicates the amount of time that a user needs to wait for its buffer to be fulfilled. The results confirms with the real experiences from users that on average a user needs to wait 10-20 seconds before it can start playing the video program.

Fig. 7 shows the media player ready time distribution under four different time periods: (i) 01:00 to 13:29, (ii) 13:30 to 17:29, (iii) 17:30 to 20:29, and period (iv) 20:30 to 23:59. Observed from the figure, we can clearly see that the media ready time is considerably longer during period (iii) when the join rate is higher.

The rationale behind the above observations is how the overlay is constructed and maintained in the Coolstreaming. Specifically, each peer maintains a partial view of the overlay through the mCache. The update of the mCache entries is achieved by randomly replacing entries when new partnership is established. Older peers or less active

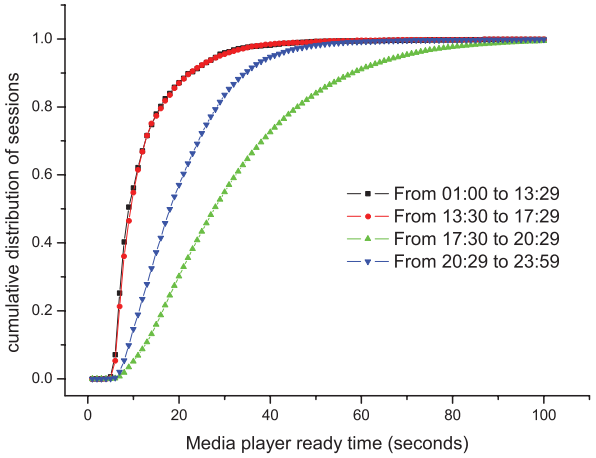

Fig. 7. Distribution of media player ready time under four different time periods

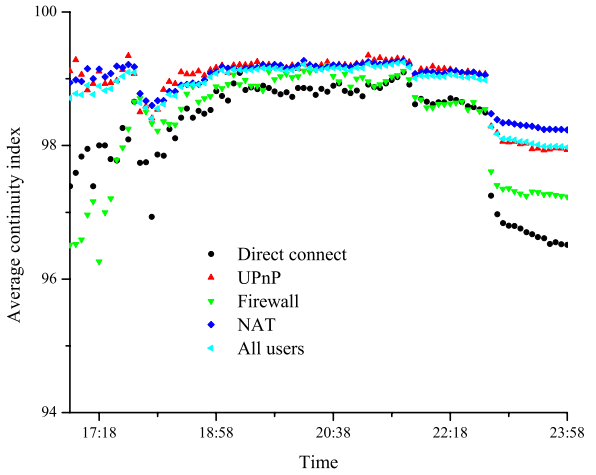

Fig. 8. Average continuity index against time with different user connection types

peers will thus be removed from the mCache gradually. However, during flash crowds, the mCache might be filled with too many newly joined peers, which often cannot provide stable video streams. Under such a replication algorithm, it takes longer time for a newly joined peer to obtain video stream.

Possible improvement can be done by designing a more effective mCache replication algorithm that enables the mCache to converge to more stable peers rather than newly joined peers. Peers will then have more chances to establish stable partnership for retrieving the video content.

D. Quality of Service

In this subsection, we examine the main QoS related measurements. Continuity index is defined as the number of blocks that arrive before playback deadlines over the total number of blocks [3]. Fig. 8 plots the average continuity index against time for different types of users in the system during peak hours. This basically captures the video playback quality for normal sessions. There are several observations from this figure. First, all type of users experience very high continuity index (consistently over 98%). In other words, there is nearly no disruption once a user is starting viewing a video program, confirmed by the real user experiences. The decrease of the continuity index after 22:30 is due to the program ending, as users start to leave the system.

Perhaps the most interesting observation is that the continuity index of direct-connected users is slightly lower than that of NAT or firewall users. We believe this could be caused by the churn effect. Observed from Fig. 5b that there is a large number of users joining and leaving the system, during which the small percentage of the direct-connected users are swamped by a large number of partnership establishments and stream requests.

This will congest the direct-connected users, it will take t_{lose} to one of the children to give up due to the sub-stream lagging. The nodes can still receive blocks from those temporary parents during the time interval $t_{\downarrow}$. On the other hand, the catch-up processes of users behind NATs or firewalls are often too slow and they will simply depart and re-enter the overlay during peer churns, resulting (i) the low continuity indices of NAT or firewall users could not be reported to the log server due to their departures and the $5-$minute granularity of state report; (ii) re-entering nodes are treated as newly joined nodes and experience catch up processes before the media player ready events. The long response time and zero continuity index during the catch-up process will not be reflected to the performance measurement, since continuity indices are only reported by state reports. Consequently, the average continuity indices of NAT or firewall users can be higher than (but unrealistic) than average continuity index of direct-connect users. However, this does not seem to pose serious problems, as the difference of the continuity index is marginal.

E. Scalability

Scalability is generally considered as one of the inherent characteristics in a P2P system. This is made possibly by each participating peer's actively contributing resources. In a macro-view of such a system, it is generally believed that more participants increase the likelihood for each peer node to retrieve the content from one another. However, a closer examination reveals that this is not always the case given that each newly joined participant demands certain level of service and also competes for available yet finite resources. In BitTorrent-like file sharing applications, the integrity of a file is the only metric of relevance. It is not surprising to see that some time it takes hours even days for a user to successfully download a file. We believe that this tolerance in time plays an important role in the scalable property of BitTorrent-like applications. From the model in [22], the download time of users perceived also depends heavily on the number of seeds within the system. It is common in BitTorrent-like applications that peers stay in the system even after the completion of download, thus act as seeds for others.

P2P live video streaming applications possess signifi-

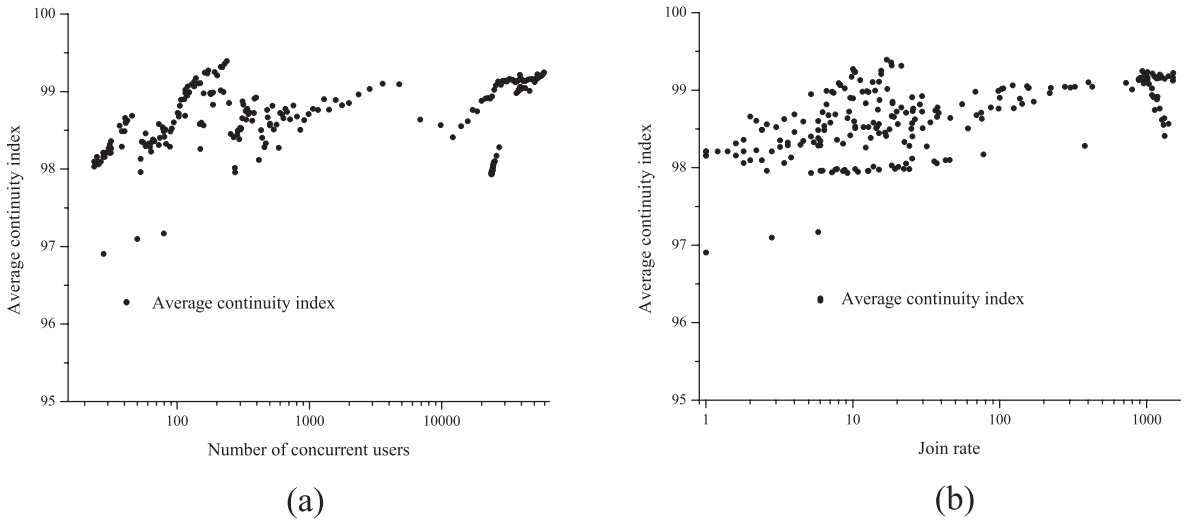

Fig. 9. (a) The average continuity index against system size; (b) The average continuity index against join rate

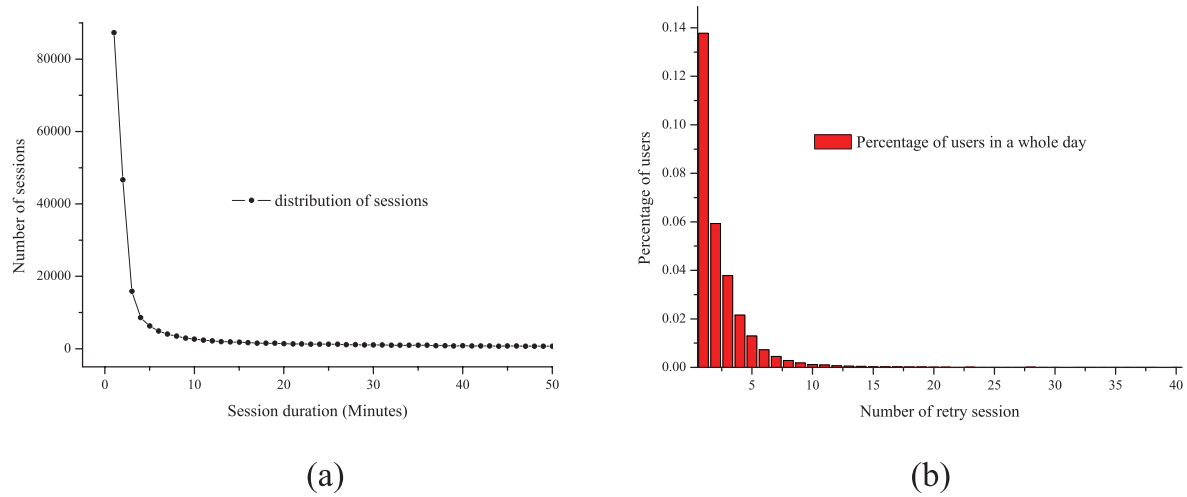

Fig. 10. (a) The distribution of session duration; (b) The distribution of re-try sessions

cant difference in service requirement and user behavior. The quality of service is driven by the stream continuity and timing requirement, which essentially invalidate the trade-off of the delay for the scale exhibited in P2P file applications. As discussed earlier, the existence of direct-connect and UPnP peers play an important role in supporting large number of peers in a P2P streaming system. This was also investigated in [23] that pointed out that there exists a critical value in the ratio of the number of high upload contribution peers and the number of opposite peers in the system.

Fig. 9a and 9b plot the average continuity index against system size and join rate in a particular day. This shows that the system achieves excellent scalability as observed from both figures, in which large number of users can be accommodated by the system under different system size and burst arrivals (often referred as flash crowd); at the same time, the continuity index in the system can maintain at very high level (around 97%).

The above discussion, however, can be misleading, since this only counts for normal sessions, i.e., the sessions that have successfully started video program. Fig. 10a plots the distribution of the session duration. On one hand, the heavy-tailed distribution indicates that once the user can successfully obtain the video stream, they are fairly stable and remain in the system throughout the entire program duration. On the other hand, this also shows that there are a significant number of short sessions (less than 1 minute). This implies that many users initiate joining multiple times before successfully obtaining the video program. This is consistent with the results obtained earlier [3], [20] that in such a system with a random partnership algorithm and high percentage of nodes behind NAT or firewall, it often takes longer time for a peer to locate capable streaming partner(s). This behavior is better illustrated in Fig. 10b, which shows the number of re-trials for all users. This indicates that 20% of the users have tried 1 or 2 times in order to obtain successful video session. This demonstrates that flash crowd has significant impact on the initial joining in a P2P streaming system.

Analyzing the scalability in a P2P streaming system

is complex which requires further examination. The argument is that the simple notion of the scalability only refers to the number of users that a system accommodates; this, however, is inadequate in a P2P streaming system as this fails to capture the timing requirement mandated by applications. Generally speaking, we believe that there are three factors that influence the scale of a P2P streaming system, system capacity, latency and overlay stability. The system capacity not only refers to the aggregate upload bandwidth in the system, but also reflects the number of peers that can be supported. Latency influences the stream continuity and overlay stability relates to the service disruption.

VI. Conclusion

In this paper, we closely examined the design principles, system dynamics and performance in Coolstreaming system. This demonstrated practically that a simple random peer selection algorithm coupled with multiple sub-stream transmission technique has the potential to scale. We believe that a working system is essential in understanding the fundamental design trade-off in P2P streaming systems. By leveraging a set of real traces obtained from live event broadcasts, we studied the workload, system dynamics and performance measurement in a P2P live streaming system. We showed 1) there is highly unbalanced distribution in term of uploading contributions from peer nodes, which we believe has significant implications on the resource provisioning in the system; 2) the results indicated that the overlay topology can lead to convergence with the self-evolving property from each participating peer; 3) the sub-stream and diversity of content delivery can minimize the disruption of video playback.

While the results from this study are encouraging, there are several open issues that need further examinations. First, the data set does not allow us to derive the peer-wise performance, which we believe it is of great relevance in understanding the self-stabilizing property of the system. Second, it is important to analyze the resource distribution and bottleneck in the system. Third, optimizations can be explored in content delivery and buffer management for performance enhancement.

References

[1] http://www.techweb.com/wire/networking/183700547

[2] http://www.thewhir.com/marketwatch/aol070505.cfm

[3] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "DONet/Coolstreaming: A Data-driven Overlay Network for Live Media Streaming," in *Proc. of IEEE Infocom*, March 2005.

[4] http://www.pplive.com

[5] http://www.sopcast.org

[6] http://www.ppstream.com

[7] X. Hei, C. Liang, J. Liang, Y. Liu and K. W. Ross, "Insights into PPLive: A measurement study of a large-scale P2P IPTV system," in *Workshop on Internet Protocol TV (IPTV) Services over World Wide Web in conjunction with WWW2006*, Edinburgh, Scotland, May 2006.

[8] A. Ali, A. Mathur and H. Zhang, "Measurement of Commercial Peer-to-Peer Live Video Streaming", *Workshop in Recent Advances in Peer-to-Peer Streaming*, August, 2006.

[9] S. Deering, "Multicast Routing in Internetworks and Extended LANs," in *Proc. of ACM Sigcomm*, August 1988.

[10] P. Francis, "Yoid: Extending the Internet Multicast Architecture," http://www.icir.org/yoid

[11] Y. Chu, S. G. Rao and H. Zhang, "A Case for End System Multicast," in *Proc. of ACM Sigmetrics*, June 2000.

[12] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek and J. W. O. Jr., "Overcast: Reliable Multicasting with an overlay Network," in *Proc. of OSDI*, October 2000.

[13] 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*, October 2003.

[14] V. N. Padmanabhan, H. J. Wang, P. A. Chou and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking", in *Proc. of NOSSDAV*, May 2002.

[15] V. Venkararaman, K. Yoshida and P. Fancis, "Chunkspread: Heterogeneous Unstructured End System Multicast," in *Proc. of IEEE Infocom*, April 2006.

[16] N. Magharei and R. Rejaie, "On Design and Evaluation of P2P Mesh-based Streaming," in *Proc. of IEEE Infocom*, May 2007.

[17] T. Silverston, and O. Fourmaux, "P2P IPTV measurement: A comparison study," *Tech. Rep.*, October 2006. http://www.arxiv.org/abs/cs.NI/0610133

[18] C. Wu, B. Li, S. Zhao, "Magellan: Charting Large-Scale Peer-to-Peer Live Streaming Topologies," to appear in the *Proc. of 27th International Conference on Distributed Computing Systems*, Toronto, June 25-29, 2007.

[19] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt, "Mapping the PPLive network: Studying the impacts of media streaming on P2P overlays," *Tech. Rep.* of University of Illinois at Urbana, August 2006.

[20] X.-Y. Zhang, J. Liu, and B. Li, "On Large-Scale Peer-to-Peer Live Video Distribution: Coolstreaming and Its Preliminary Experimental Results," in *Proc. of IEEE MMSP'2005*, October 2005.

[21] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie, "Peer-to-peer membership management for gossip-based protocols," in *IEEE Transactions on Computers*, Vol. 52, No. 2, February 2003.

[22] D. Qiu, R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Proc. of ACM SIGCOMM*, August 30- September 3, 2004.

[23] R. Kumar, Y. Liu, and K.W. Ross, "Stochastic Fluid Theory for P2P Streaming Systems," in *Proc. of IEEE Infocom*, May 2007.