An Empirical Study of the Coolstreaming+ System

Bo Li, Susu Xie, Gabriel Y. Keung, Jiangchuan Liu, Ion Stoica, Hui Zhang, and Xinyan Zhang

***Abstract*—In recent years, there has been significant interest in adopting the Peer-to-Peer (P2P) technology for Internet live video streaming. There are primarily two reasons behind this development: the elimination of infrastructure support and the self-scaling property of P2P systems. The success of our system Coolstreaming represented one of the earliest large-scale P2P video streaming experiments. Since then, there have been several large-scale commercial deployments. With desirable content, these systems have the potential to scale orders of magnitude beyond the existing academic P2P prototypes. However, to transform this potential into reality, we need to understand the key design trade-offs and principles, as well as the design limitations of these systems.**

There are two main design decisions in a P2P streaming system: (i) How to form an overlay, and (ii) How to deliver the content. Coolstreaming adopts a gossiping protocol for overlay construction, and a swarm-based protocol for content delivery. While these protocols provide excellent flexibility and effectiveness in dealing with system dynamics and random failure, their impact on the performance and the system scalability remain less known. This paper takes an inside look at a commercial system based on the Coolstreaming, called Coolstreaming+. We explore its design choices and the impact of these choices on streaming performance. Specifically, by using internal traces generated by recent live broadcast events, we study the workload, performance, and dynamics of the system. Based on these traces, we show that (1) the churn is the most critical factor that affects the overall performance of the system, and (2) there is a highly skew resource distribution in P2P streaming systems, which has significant impact on resource allocation. We further discuss the impact of these observations on the system properties, and present solutions to deal with various design challenges. In particular, we suggest solutions to deal with the excessive start-up time and high failure rates during flash crowd, which are two of the main challenges any streaming system needs to address.

***Index Terms*—Video streaming, peer-to-peer technology, measurement, IPTV.**

I. Introduction

RECENTLY there has been significant interest in adopting the Peer-to-Peer (P2P) technology for Internet live video streaming [1]. There are primarily two factors fueling this development: First, a P2P system requires no or minimum support from specific network infrastructure supporting (like IP multicast), and thus is cost-effective and easy to deploy. Second, in such a system, each 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 [2] represented one of the earliest large-scale peer-to-peer video streaming experiments, 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 the first release of Coolstreaming, while keeping the basic random partner selection, we have enhanced the system in nearly all aspects, specifically: 1) the initial system adopted a simple *pull-based* scheme for content delivery based on content availability information. This incurs per block overhead, and more importantly, it results in a longer delay in retrieving the video content. We have now implemented a *hybrid pull and push* mechanism, in which the video content is pushed by a parent node to a child node except for the first block. It remarkably lowers the overhead associated with each video block transmission, reduces the initial delay and increases the video playback quality; 2) a novel multiple sub-stream scheme is implemented, which essentially enables multi-source and multi-path delivery for video streams. Observed from the results, this not only enhances the video playback quality and but also improves the effectiveness against system dynamics; and 3) the buffer management and scheduling schemes are completely re-designed to deal with the dissemination of multiple sub-streams.

In the mean time, Coolstreaming-inspired commercial systems such as PPlive [3] and Sopcast [4] have also been developed and deployed. however, question remains whether a P2P streaming system can really scale to the capacity of commercial TV channels, which can be millions or even billions. 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 constraint in prior studies was the lack of internal knowledge in the architec-

Manuscript received March 16, 2007; revised August 29, 2007. The research was supported 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.

B. Li, S. Xie, and G. Keung are with the Department of Computer Science, Hong Kong University of Science and Technology (e-mail: {bli, xiesusu, gabriel}@cse.ust.hk).

J. Liu is with th School of Computing Science, Simon Fraser University.

I. Stoica is with the Computer Science Division, University of California at Berkeley.

H. Zhang is with the Computer Science Department, Carnegie Mellon University.

X. Zhang is with Roxbeam Corp. Beijing, China.

Digital Object Identifier 10.1109/JSAC.2007.071203.

ture and control mechanisms, due mainly to the proprietary nature of commercial systems. Existing measurement studies [5][6] 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 is based on an internal logging system that we have designed with full knowledge of the Coolstreaming architecture and control mechanisms. Specifically, we leverage a large set of traces obtained from the very recent live streaming events using Coolstreaming; we study the workload, performance, and dynamics of the system. There are three major observations from this empirical study, 1) there is a highly skew resource distribution in the systems; 2) the performance is mostly affected by the system dynamics, in particular the churn; and 3) the excessive start-up time and high failure rates during flash crowd remain critical problems. We further study their implications and the causes from the internal system design, in particular 1) We describe the fundamental design principles and trade-offs in Coolstreaming system; 2) We discuss how a random peer selection algorithm can lead to the topology convergence; and 3) using the set of real traces, we quantitatively demonstrate how a novel buffering technique resolve the problems associated with dynamics and heterogeneity.

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 investigates the system dynamics, in particular peer joining and peer adaptation processes. Section V discusses system configuration, log system and sampling errors. Section VI presents results from live event broadcast and examines the performance. Finally, Section VII concludes the paper with highlights on future research.

II. Related Work

Earlier Internet video streaming systems were largely built on the *IP multicast* model [7]. 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 "stateless" 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 [8]-[10], 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 [9][10] or multi-trees [11][12]. 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 [2], PPlive [3], and Sopcast [4], 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 [13] , 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 [14], 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 [5], 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 [6] 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 [5][15][16]. 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.

III. Basic Architecture

Coolstreaming [2] was developed in Python in earlier 2004. Its implementation is platform independent and 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 400 Kbps, with users from 24 different countries. More details can be found in [2][17].

The effectiveness of the design principle and architecture of Coolstreaming 0.9 have been demonstrated successfully and are thus largely kept in the later development of Coolstreaming. In this section, we present an overview of the basic design as well as the most recent updates.

A. Basic Components

The system consists of *five* basic modules: 1) the Membership manager, which maintains the partial view of the overlay; 2) the Partnership manager, which establishes and maintains partnership with other nodes; 3) the Scheduler, which is responsible to schedule data transmission across streams; 4) the Buffer, which stores video data before playback; 5) the Buffer Map, which represents the current status of the buffer and data requests.

B. Overlay Construction and Maintenance

In Coolstreaming, each node has a unique identifier and maintains a membership cache (*mCache*) containing a partial list of the currently active nodes in the system. A newly joined node contacts the boot-strap node for a list of nodes and stores that in its own mCache. It then randomly selects a few nodes to establish *partnership* maintained by the partnership manager module. 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. Partnerships can be broken due to many reasons and the nodes will need to perform the *partner re-selection* to maintain the continuity of video streams.

C. Sub-streams and Buffering

In the current Coolstreaming, a video stream is divided into multiple *sub-streams* and a node could subscribe for sub-streams from different partners. A 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, which consists of two parts: *synchronization buffer* and *cache buffer*. A received block is first put into the synchronization buffer for the corresponding sub-stream. They will be combined into one stream when blocks with continuous sequence numbers have been received from each sub-stream.

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.

D. Content Delivery

In a tree-based approach, content dissemination is determined by the overlay topology, in which a node typically pushes the content through the overlay topology. In the data-driven Coolstreaming, this is more complex and often requires: 1) peers gossip by exchanging content availability information and 2) peers then use a *pull-based* approach to retrieve the content from each other. This works well for file downloading applications like BitTorrent, and was also adopted by the original Coolstreaming. There are however two disadvantages of a pure pull-based approach: (1) high bandwidth overhead and (2) long delay. These disadvantages are due to the gossiping protocol used to exchange information between peers, and that a node needs to to independently request each data block.

The current Coolstreaming adopts a *hybrid push-pull* approach, in which each peer only sends a single request for a sub-stream. Once the request is accepted, the parent node will push all subsequent data blocks from this sub-stream to the requested children peers. Similar approach has been adopted in [18]. In this hybrid approach, two parameters need to be determined: 1) from which part of the video stream should a newly joined node start to subscribe to. In other words, it needs to determine the initial sequence number of the block that the node will start to retrieve; 2) how to select an appropriate partner to subscribe for each sub-stream once the initial sequence number is determined. We will further elaborate on this as well as compare the hybrid approach with pure pull operations in the following sections.

TABLE I
SYSTEM PARAMETERS OF COOLSTREAMING

R	stream bit rate
K	number of sub-streams
B	buffer length (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 other partners and parents
T_a	period for a peer to re-select a parent if needed
D_p	sub-stream degree of node p
$t_{\downarrow}$	a nodes can still receive blocks from those temporary parents during the time interval
t_{lose}	time for one of the children to lose the competition due to a subscribed sub-stream lagging behind others

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 Join

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 sub-section) 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

Due to congestion and churns, it is important for any P2P system to do *peer adaptation*, i.e., to search for new parent(s). In the Coolstreaming system, this is triggered either by the insufficient bandwidth received from a parent, or the existence of more capable partners. To facilitate adaptation, 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 is 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 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. 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.

When a peer selects a new parent from its partners, 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 that accepts the new selection request will then push out all blocks of a sub-stream in need to the requesting node. It is possible that a parent with insufficient uploading capacity is selected, in which case all children nodes have to compete for the insufficiently aggregated upload capacity, and eventually one or more nodes will lose and trigger the peer adaptation. We call this situation *peer competition*, which will clearly cause a chain reaction that disrupts streaming and makes the overlay unstable. To address this problem, we introduce a *cool-down timer* that confines nodes to perform peer adaptation once only within a cool-down period of time T_a. As such, a node can locate a capable parent only within a certain time limit.

V. Log and Data Collection

In this section, we describe the system configuration, log system, format and sampling errors.

A. System Configuration

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, as shown in Fig. 1. 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.

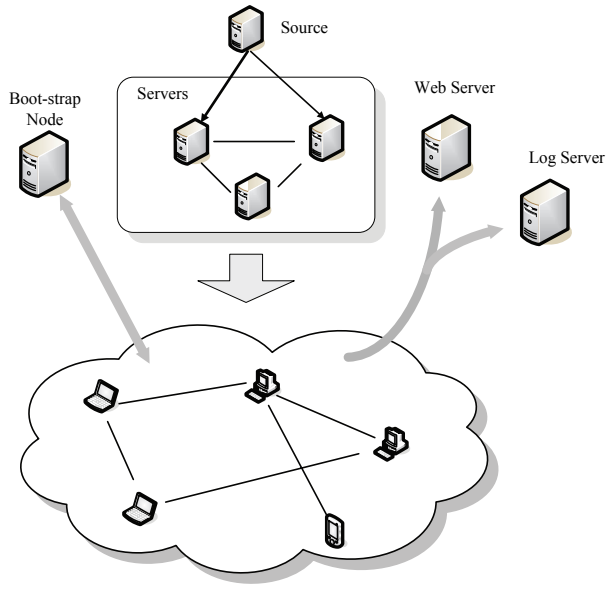


Fig. 1. Coolstreaming System Configuration

B. Log System and Log File

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 com-

pact report that records a series of activities to reduce log server's load.

C. Sampling Errors

Given the asynchronous nature of the system, the sample errors are inevitable. We will next discuss the main factors causing the errors and their impacts on the results.

Clock skew: There are two time values for each report. The first is the time when the report is generated at a peer, and the second is the time when the report is received by the log server. We refer to them as *generation time* and *receive time*, respectively. The generation time is appended by peers with the local time and the receive time is appended by the log server. Since clocks between peers and the log server might not be synchronized with each other, this potentially causes measurement inaccuracies. However, there are several reasons that these are not as severe as they appear to be: 1) By default, the MS Windows automatically synchronize with some Internet time servers; 2) The server time can always be used as a reference time whenever necessary; 3) Many measurements depend only on the differences from the local time, for instance the duration of a video session, which do not require synchronization.

Log server overloading: The log server can become a potential bottleneck in the system. This causes two problems: 1) It takes an excessively amount of time to respond to a report during peak times; and 2) If we use the receive time to calculate the performance metrics such as *join* and *leave rates* (defined in next section), the curve will be distorted due to the time shift.

Errors in determining connection type: There is a need to classify user connection types in order to determine the *uploading capacity distribution* (see Section VI). 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. It is possible that later an incoming partnership is established, and so the peer will report the change of the connection type.

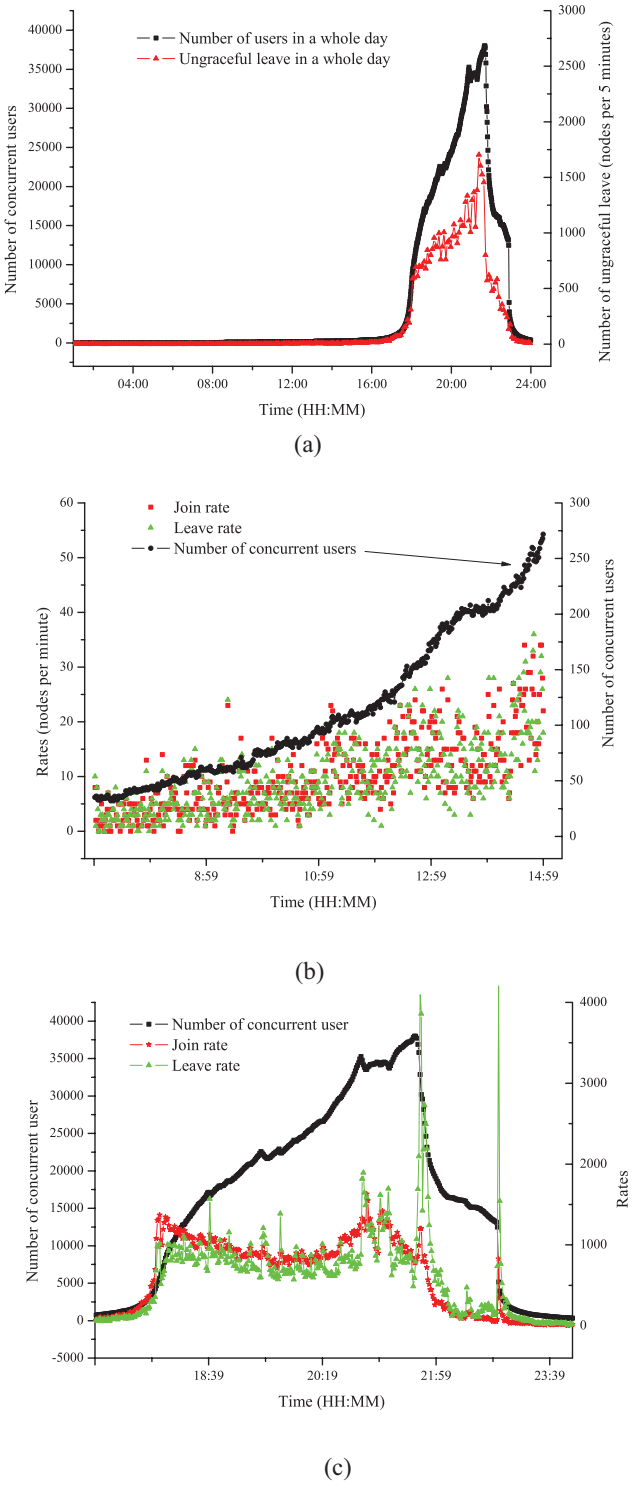


Fig. 2. Evolution of the number of users in the system: (a) in the whole day; (b) from 07:00 to 14:59 ; (c) from 18:00 to 23:59

This implicates a longer duration time can increase the accuracy in determining peer connection type. After the first state report, most of peers should be categorized into correct type after first five minutes. The cases in which we fail to determine the connection type involve usually short sessions. Furthermore, a larger maximum number of partners also can increase the accuracy in determining peer connection type.

Ungraceful leave: A user can leave the system anytime, and its ActiveX control may not have sufficient time to

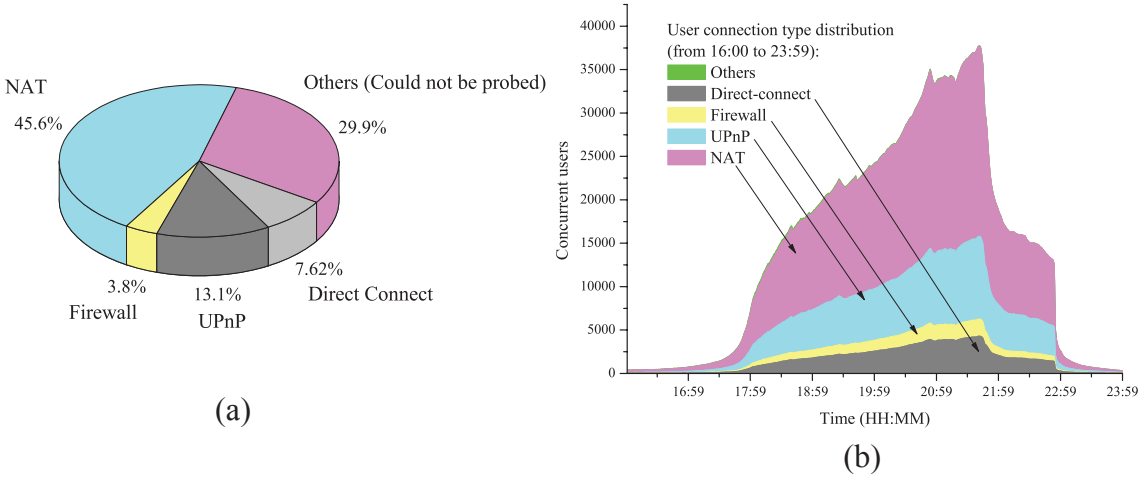


Fig. 3. (a) User types in a whole day; (b) User types from 18:00 to 23:59

initiate a "graceful" leave. For example, when an impatient user closes the browser directly, the ActiveX control will be killed immediately without reporting to the log server or notifying its partners. In that case, the peer's partners will not be able to report any activities for this peer in the next report. Such ungraceful leaves are often short and happen at peak hours, and bring discrepancy in counting the number of video sessions. Ungraceful leaves can be detected by checking the existence of periodical state reports from peers. If a peer missed two consecutive state reports, we treat it as an ungraceful departure. Given the fact that the report period is 5 minutes, it may take us up to 10 minutes to detect a ungraceful departure. However, the ungraceful leaves represents only a small percentage of the total leaves. In our measurements, we have found 15991 ungraceful leaves, which account for only 5% of the total sessions. To eliminate the bias introduced by ungraceful leaves, we have filtered out these leaves from the results we present in Section VI. The only exception is Fig. 2a, which shows the number of sessions over time.

Inaccurate view of overlay: The log system receives *partnership report* every 5 minutes. Given this coarse granularity and synchronization problem in the reports, this affects the accuracy in deriving the overlay at an instant of time. Further, the current log system does not record any pair-wise performance measurements, making it difficult to obtain the overlay and partnership information.

The data set does not allow us to derive accurate overlay topology. We believe in future, it is of great relevance to examine the concrete evidences about how the overlay topology migrates and self-stabilizes under the random topology construction.

VI. Results

We now present and analyze the results obtained from live broadcast events of Coolstreaming on 27^{th} September, 2006. This is a typical date that involves both steady group of users as well as flash crowd, and we have observed similar results in other measurement periods. We will describe the user and session behaviors, and analyze the fundamental performance limitations as well as impacts from the system design. We will also examine the resource distribution in the system and the QoS related metrics.

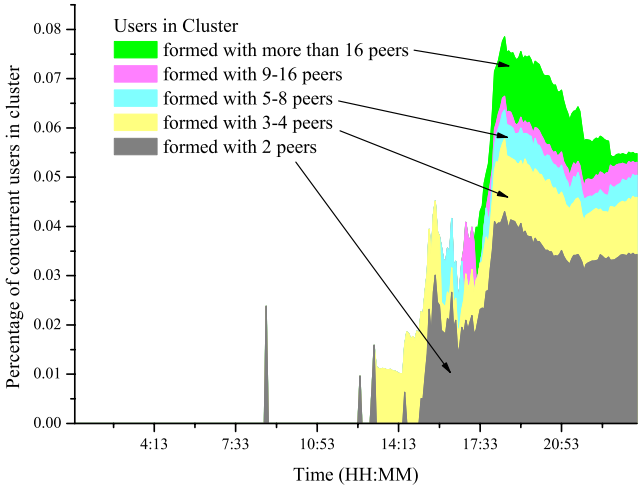


Fig. 4. Number of cluster formed with multiple peers with different user connection types

A. User Types and Distribution

Upon initiating a join operation, each user is assigned a unique ID. The log system keeps track of the user IDs as well as their IP addresses and port numbers. The *total number* metric represents the number of users with unique IDs that are in the system at a particular time. A *cluster* represents a set of users having the same IP addresses but distinctive IDs. As we will show later, typically a clusters represent a set of users behind the same NAT. Finally, a *session* denotes the time interval between the time the user joins the system until it leaves. In each session, a 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 bootstrap node;
- *Start subscription* event: This event is reported as soon as the client establishes partnership relations with other

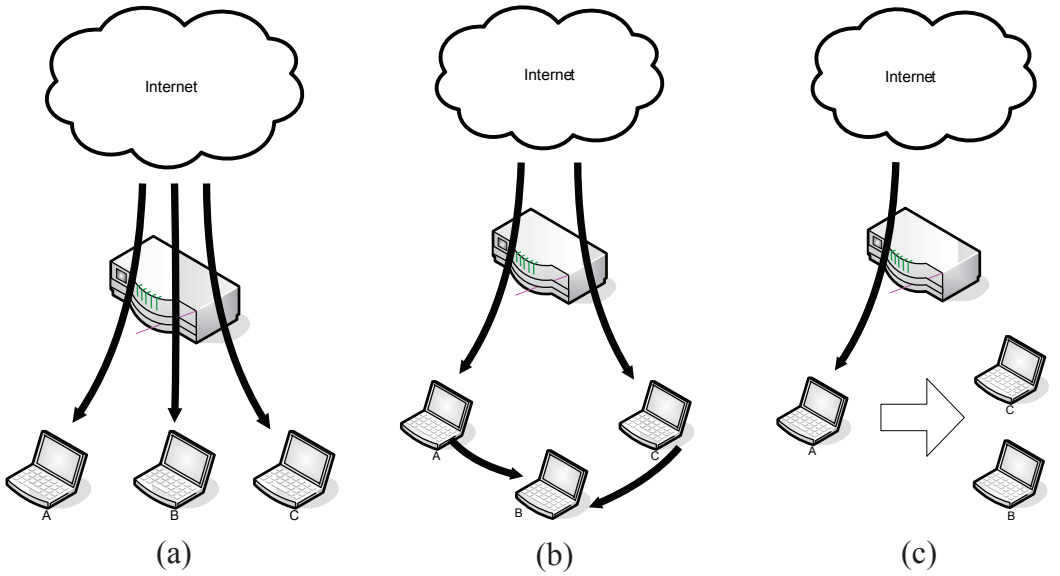


Fig. 5. (a) Peers behind the same NAT device; (b) Overlay construction improvement for NAT cluster; (c) Another optimization for NAT cluster

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.

As defined earlier, a partnership represents the relationship between two users, which may exchange data availability information. An *outgoing* partner is a partner that can only initiate partnership establishment; another node cannot initiate partnership establishment to an outgoing partner. Outgoing partners are typically the nodes behind NATs and firewalls which cannot be contacted directly from the public Internet. 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.

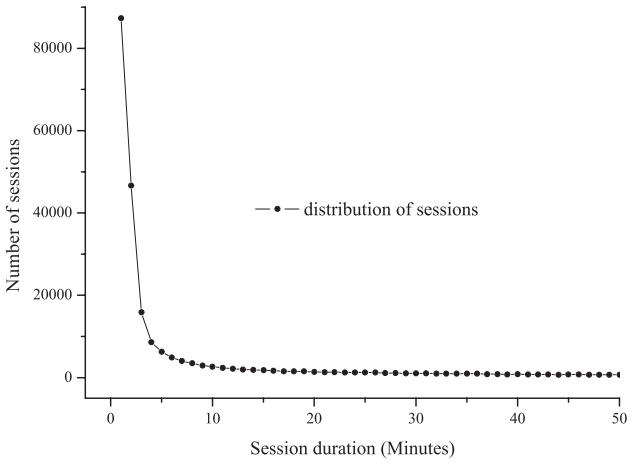
Fig. 2a plots the number of sessions in the system in the whole day, as well as the number of ungraceful leaves versus time. While the number of ungraceful leaves increases during peak hours, it never exceeds 5% of the number of sessions in any time periods. The result shows the number of ungraceful leaves only occupies small portion of the whole population, so that it should have small influences to the performance measurements. Fig. 2b and Fig. 2c plot the number of sessions in the system, join and leave rates, with a granularity of 1 minute between 07:00-14:59 and 18:00-23:59, respectively. These plots illustrate user behavior in a typical weekday and during the peak hours, respectively. The number of concurrent users and the join rate are relatively modest during 07:00-14:59, after which, they start to increase significantly and peak around 22:00. The sudden drop in the number of users around 22:00 is caused by the ending of some programs. This shows the scalability of the system, as the the number of nodes in the system can quickly ramp up to $40,000$.

Fig. 3a and Fig. 3b plot the different types of users in the system for the whole day and between 18:00-23:59. There are three interesting observations: First, only around 20% of users are directly connected or are connected via UPnP. Thus, only a small percentage of peers in the system can be directly accessed. This is consistent with the measurements in [2][17]. Second, there is a significant number of NAT users (more than 45%) with limited uploading capabilities. Third, there are some users that we cannot determine their connection types in time. This is because either the delay in a log report when the log server is busy, or a partnership is not stabilized before we can determine the connection type.

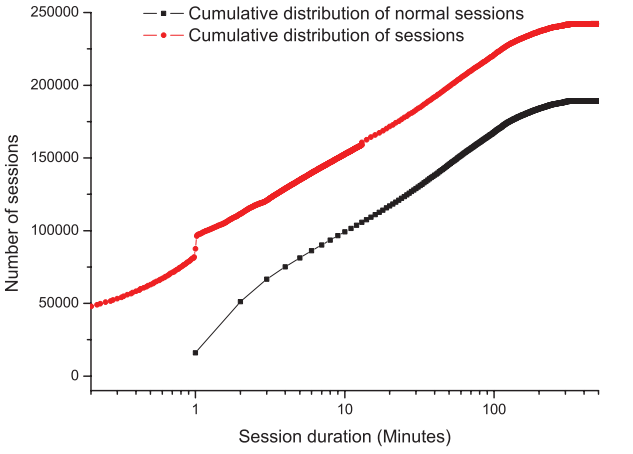
Fig. 4 summarizes the peer clustering in the overlay, mainly for users behind NATs. As expected, this has a negative impact on the peer bandwidth contribution. Fig. 5 illustrates this problem. In particular, peers behind the same NAT device compete for incoming bandwidth at the device. As depicted in Fig. 5a, all the three peers share the incoming bandwidth of the edge device from outside peers (with public IP address). Furthermore, the data received by each peer cannot be directly shared with each others. Therefore locality can be possibly explored to improve the content delivery, specifically: (i) Peers at the edge should be able to negotiate with one another to alleviate the redundancy. As shown in Fig. 5b, the incoming traffic can be kept at a reasonable level under the upper bound of the edge device, while the leave of the one peer, e.g. A or C, shall not cause interruption to all other nodes; (ii) Another improvement is possibly made by exploring the broadcast nature of local network channels. For example, in Fig. 5c, peer A can broadcast its data in the LAN to reduce the burden at the edge devices.

B. Session Distribution

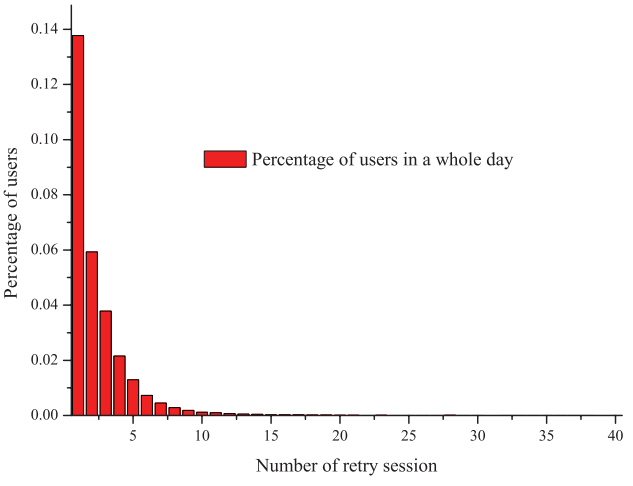
Recall that a *session* represents a pair of join/leave event of a user. The *session duration* is the time interval between a user joining and leaving the system. For a *normal session*, the events reported include: 1) join event, 2) start subscription event, 3) media player ready event, and 4) leave event. The *media player ready time* is the time between a pair of join and media player ready events; the *start subscription time* is the time between a pair of join and start subscription events.



(a)



(b)



(c)

Fig. 6. (a) Distribution of session duration; (b) Cumulative distribution of session duration and normal session duration; (c) Distribution of re-try sessions

In a normal session, the *average upload/download bit rate* of a user can be calculated by dividing the total number of bytes (sent/received) by the session duration. A *failed session* is a session for which a leave event is reported before the media player ready event. This indicates that a user fails to start playing the video. The *failure rate* is computed by the total number of failures every minute. The failures could be counted more than once if a user re-joins the system after it leaves.

Fig. 6a plots the distribution of the session duration. The result suggests that once the users 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 plot also shows that there is a significant number of short sessions. A comparison is shown in Fig. 6b, which plots the cumulative distribution of session duration and normal session duration. The figure describes most of short sessions belong to start-up failures of newly joined nodes. This is consistent with the results obtained earlier in [2][17]. We believe this is caused by the combination of the random partnership algorithm and high percentage of nodes behind NATs or/and firewalls. As illustrated in Fig. 6c, over 20% of the users have tried 1 or 2 times in order to successfully start a video session. Hence, a flash crowd has significant impact on the initial joining phase in a P2P streaming system.

C. System Dynamics

We next examine more closely the performance impact from system dynamics. Fig. 7a and Fig. 7b plot the join rate, leave rate, and failure rate every minute before 21:00. It clearly demonstrates that there exists a strong correlation between failure events and join events, and failure events and leave events. Fig. 8 plots the failure rate versus the number of users in the system. This figure shows plots the failure rate versus the system size for two cases: 1) when the rate of joins/leaves is lower than 300 nodes per minute, and 2) when the rate of joins/leaves is greater than 300 nodes per minute, respectively. The circular data points cluster into two groups, which both correspond to low join, leave and failure rates. From the above observation, the clusters correspond to the steady state of the overlay. The triangular data points do not show any correlations between failure rate and system size, implying that the failure is mostly caused by churn, while not the size of the system.

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 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.

This can also be validated through the response time data. Fig. 9 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

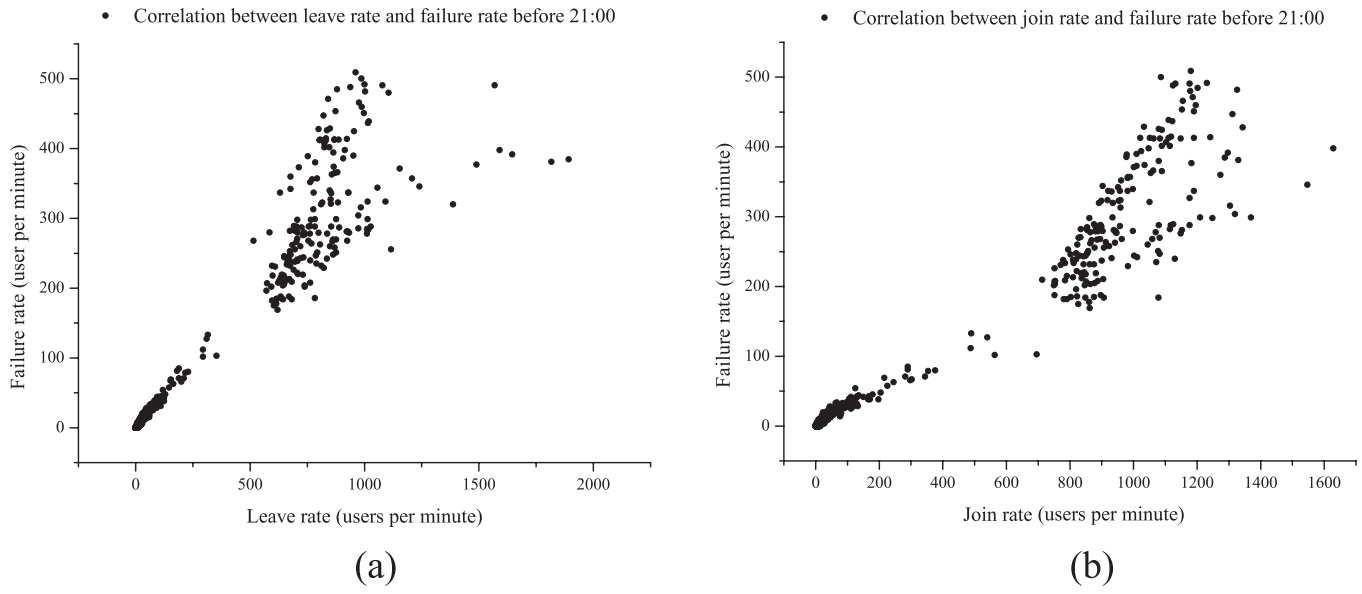


Fig. 7. (a) Correlations between join rate and failure rate before 21:00; (b) Correlations between leave rate and failure rate before 21:00

ready time is considerably longer during period (iii) when the join rate is higher. 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. Resource Distribution

In this subsection, we examine the resource distribution in the system and its implication on the performance. We define a metric called *contribution index*, which is the aggregate upload bandwidth (bytes sent) over the aggregate download bandwidth (bytes received) for each user. This is different from the *Resource Index* [19] that describes the resource available in the system. The contribution index here captures the actual uploading contribution from users during a particular broadcast event.

There are two issues relevant to the computation of contribution index. First, since the log system does not differentiate between the types of traffic, the aggregate download bandwidth is mixed up with other traffic. One discrepancy can be caused by the control traffic; for instance, two partners exchanging video availability information through BMs. Another factor is the existence of servers. When the number of users in the system is small, most users can be supported by the servers, thus the contribution index from the users is relatively low. Since the contribution index reflects the willingness of each user for sharing video content and the uploading capability, if the aggregate upload capacity from a user is zero, the contribution index is also zero. On the other hand, if the aggregate upload (bytes sent out) equals to aggregate download (bytes received) of a user, the contribution index is one, indicating that the user is capable of providing full video streams to another user. To better capture the granularity of the uploading contribution from each user, we define the contribution level categorized by the average value of the contribution index:

- *Level 0*: contribution index is larger than one. The user can upload the whole video content to another user;
- *Level 1*: contribution index is between 0.5 and 1. The user can at least upload half video content to its children;
- *Level 2*: contribution index is between $1/6$ and 0.5. The user can upload at least one sub-stream to its children;
- *Level 3*: contribution index is less than $1/6$. The user cannot upload a single sub-stream to its children.

Fig. 10a and Fig. 10b plot the contribution index distributions for the entire event and for the period between 18:00-23:59, respectively. We can see that a significantly number of users (over 60%) cannot stably contribute one sub-stream. We believe this is because many of the users are behind NAT and firewalls (see Fig. 3a). Fig. 10c plots the contribution index against system size with $5-$minute granularity. The results suggest that when the system size is small, most of the uploading capacity is provided by the servers. When the number of users starts to increase, the finite uploading capacity of the servers becomes insufficient, and the uploading capacity in the system is gradually taken over by the streaming among peers.

E. Quality-of-Service (QoS) index

Finally, we examine the QoS metrics that directly related to user experience. Fig. 11a shows that the hybrid architecture can significantly reduce the initial start-up delay, which is the media player ready time of peers, compared with the pull-based system. There are two reasons behind this improvement: first, the hybrid pull and push mechanism reduces signal overhearing associated each block transmission, which further results in a faster catch up process for newly joined nodes. Consequently this helps to reduce the initial startup delay; second, the hybrid architecture improves the efficiency in data

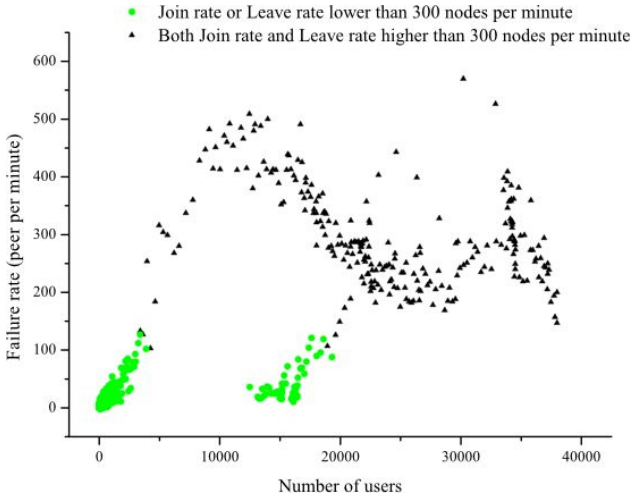


Fig. 8. Correlation between failure rate and system size

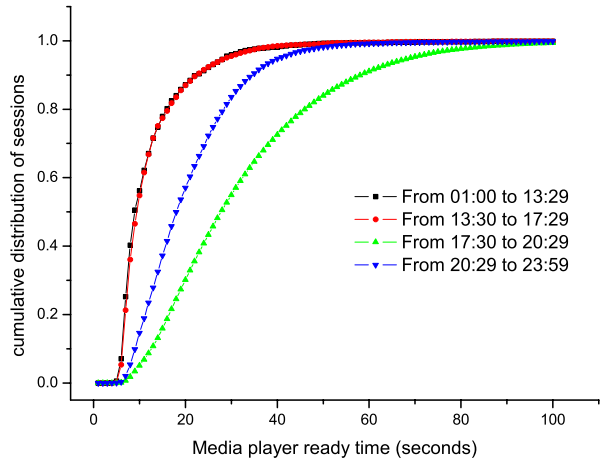
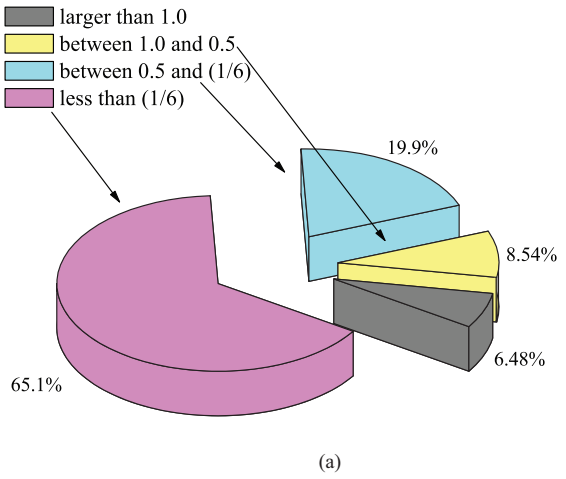
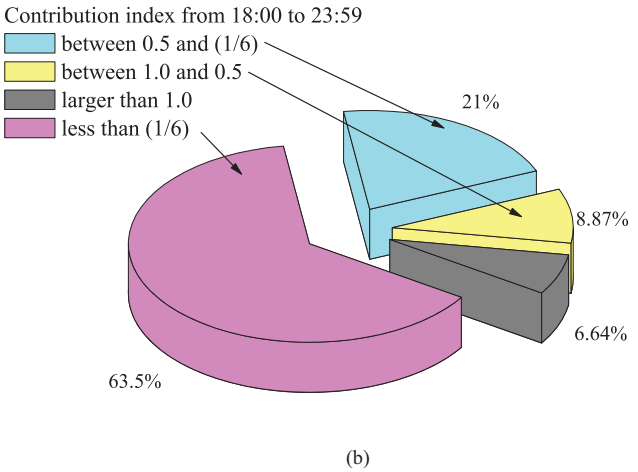


Fig. 9. Distribution of media player ready time under four different time periods





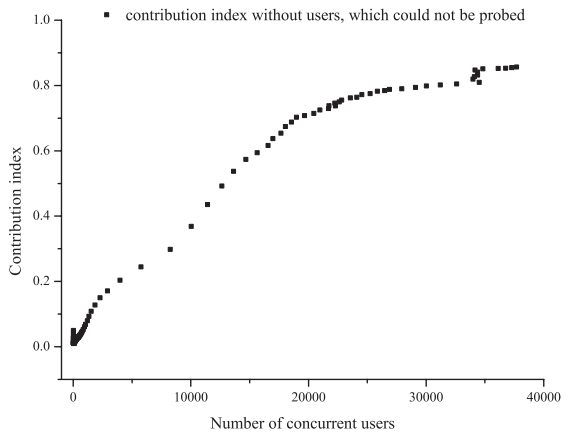


Fig. 10. (a) Distribution of contribution index; (c) Distribution of contribution index from 18:00 to 23:59; (b) Correction between contribution index and system size

transmission as the outgoing (uploading) bandwidth is more effectively utilized.

Fig. 11b 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, indicating that there is also a non-negligible amount of users that fail to find a capable parent, and thus cannot obtain the video program in time. Fig. 11b also plots the cumulative distribution of the difference between the media player ready event and start subscription event, which essentially indicates the amount of time that a user needs to wait for its buffer to be fulfilled. The results are consistent with the real life experiences that on average a user needs to wait 10-20 seconds before it can start playing the video program.

We further define the *continuity index* as the percentages of video content successfully received by each user before its playback deadline. Fig. 12a plots the average continuity index and the system size versus time, while Fig. 12b plots the average continuity index versus time for different types of users in the system. Again, all type of users experience very high continuity index. In other words, there is nearly no disruption once a user is starting viewing a video program, as confirmed by the real user experiences. There is a decrease of the continuity index after 22:30; at that time, the program ends, and the users start to leave the system.

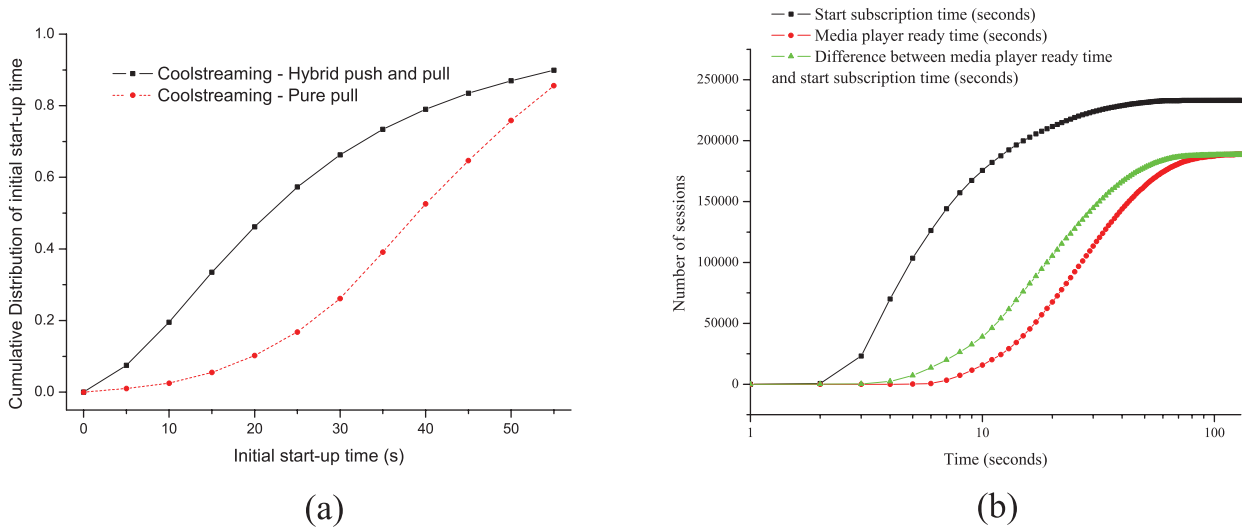


Fig. 11. (a) Comparison of initial start-up delay between pull-based and hybrid push-pull; (b) Comparison between start subscription time, media player ready time and their difference

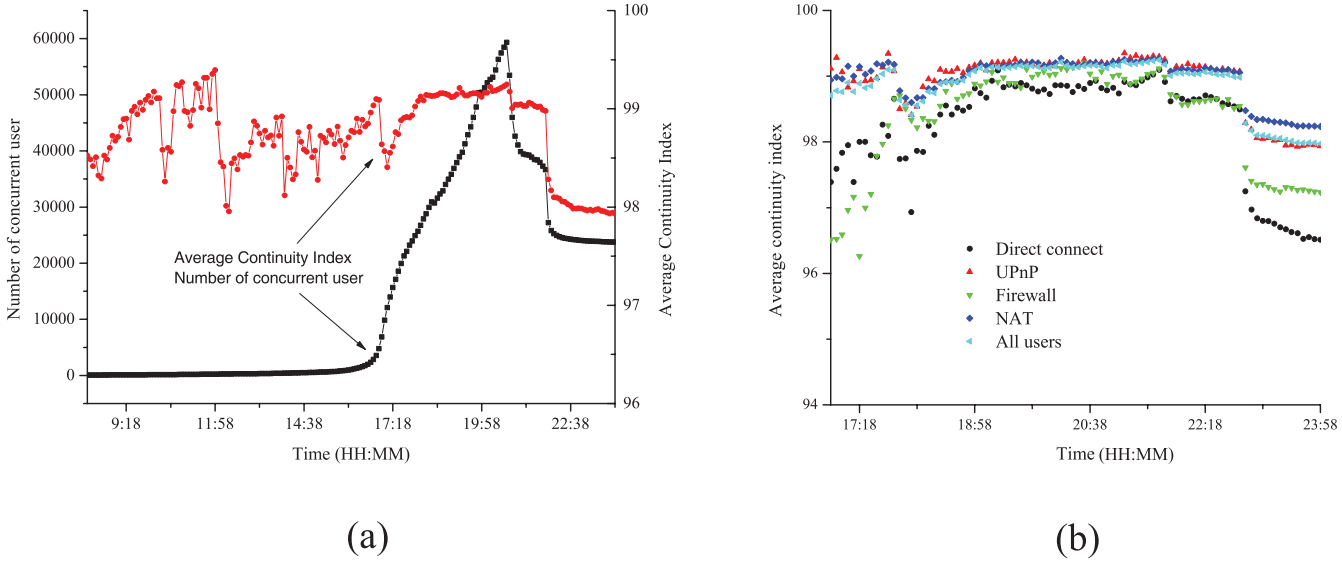


Fig. 12. (a) Comparison of continuity index and number of users ; (b) Average continuity index against time with different user connection types

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. 2c 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.

VII. Summary

In this paper, we used a large set of live streaming traces obtained from the Coolstreaming system, and studied the workload, inherent system dynamics and performance measurement. We first described the internal architecture and key design trade-off in Coolstreaming. We showed that the random partner selection and sub-streaming can effectively deal with the system dynamics. With concrete evidences, we demonstrated that: 1) The critical performance problem in P2P streaming system is the excessive start-up time and high join failure rates during flash crowd; 2) The system dynamics, in particular the churn, is the most critical factor that affects the overall performance; 3) There is a highly unbalanced distribution in term of uploading contributions from nodes, which has significant implications on the resource allocation in such a system.

We believe the lengthy start-up time and high failure rates are inherent problems in P2P streaming systems, especially with the presence of large percentages of the nodes behind NATs or firewalls. Further, when the size of the system is small, finding a capable partner can take even longer time. We have found that a certain number of server deployment is of necessity. However, pure server-based approaches like CDN can be costly and do not scale well. On top of that, there is a practical difficulty in provisioning servers on-the-fly. Hence, we believe that a large-scale commercial Internet streaming system should be a hybrid system, i.e., P2P with assistance from geographically distributed servers.

While the results from this study are encouraging and offer a number of insights in understanding the P2P streaming system, there are several open issues that need further examinations. In particular, the data set does not allow us to derive accurate overlay topology. We believe it is of great relevance to examine the concrete evidences about how the overlay topology migrates and self-stabilizes under the random topology construction. There are also many optimizations that could further improve the system performance, such as exploring the clustering behavior at different users to better utilize the resources, and to design structure-assisted content delivery mechanism to reduce the impact from system dynamics.

References

[1] J. Liu, S. G. Rao, B. Li, and H. Zhang, "Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast," *Proc. of the IEEE, Special Issue on Recent Advances in Distributed Multimedia Communications*, 2007.

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

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

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

[5] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," to appear in *IEEE Transactions on Multimedia*, 2007.

[6] 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.

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] V. Venkararaman, K. Yoshida and P. Francis, "Chunkspread: Heterogeneous Unstructured End System Multicast," in *Proc. of IEEE ICNP*, November 2006.

[14] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming," in *Proc. of IEEE Infocom*, May 2007.

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

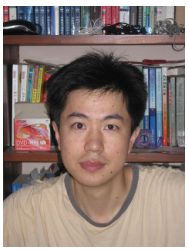
[16] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt, "Mapping the PPLive network: Studying the impacts of media streaming on P2P overlays," *Tech. Rep.* of University of Illinois at Urbana, August 2006.

[17] X. 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.

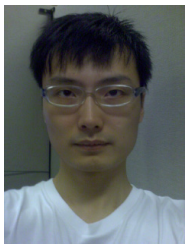
[18] M. Zhang, L. Zhao, J. L. Y. Tang, and S. Yang, "A Peer-to-Peer Network for Live Media Streaming - Using a Push-Pull Approach," in *Proc. of ACM Multimedia*, November 2005.

[19] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points," in *Proc. of ACM SIGCOMM*, August 30- September 3, 2004.

Bo Li (S89-M92-SM99) received his B. Eng. (summa cum laude) and M. Eng. degrees in the Computer Science from Tsinghua University, Beijing in 1987 and 1989, respectively, and the Ph.D. degree in the Electrical and Computer Engineering from University of Massachusetts at Amherst in 1993. Between 1993 and 1996, he worked on high performance routers and ATM switches in IBM Networking System Division, Research Triangle Park, North Carolina. Since 1996, he has been with the Department of Computer Science, Hong Kong University of Science and Technology. Since 1999, he has also held an adjunct researcher position at the Microsoft Research Asia (MSRA), Beijing, China. His current research interests are on adaptive video multicast, peer-to-peer streaming, resource management in mobile wireless systems, across layer design in multi-hop wireless networks, content distribution and replication. He is currently a Distinguished Lecturer in IEEE Communications Society. He has served on the editorial board for a large number IEEE journals such as IEEE Transactions on Mobile Computing, IEEE Transactions on Wireless Communications, IEEE Transactions on Vehicular Technology. He was the also a guest editor for three special issues of IEEE Journal on Selected Areas in Communications. He was the Co-TPC Chair for IEEE Infocom2004.



Susu Xie , (xiesusu@cse.ust.hk) received his bachelor degree from Tsinghua University in 1998. After several years experience in software development in real-time multimedia applications, he is currently a Ph.D. candidate of Hong Kong University of Science and Technology. His research interests are in the area of Internet and wireless networks with emphasis on multimedia applications.



Yik Keung , (phgab@cse.ust.hk) is currently working toward the Ph.D. degree at the Department of Computer Science and Engineering in Hong Kong University of Science and Technology. He received the B.S. and M.Phil. degree from the Department of Physics, Hong Kong University of Science and Technology in 2002 and 2004 respectively. His previous research interests are in the area of wireless communication networks, with emphasis on resource allocation and mobility management for multimedia traffic. His current research interests are in area of Internet and Peer-to-Peer networks with emphasis on multimedia applications.

Jiangchuan Liu (S01-M03) received the B.Eng degree (cum laude) from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree from The Hong Kong University of Science and Technology in 2003, both in computer science.

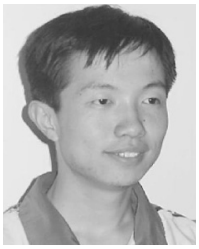
He is currently an assistant professor in the School of Computing Science, Simon Fraser University, BC, Canada, and was an assistant professor at The Chinese University of Hong Kong from 2003 to 2004. He was a recipient of Microsoft research fellowship (2000), a recipient of Hong Kong Young Scientist Award (2003), and a co-inventor of one European patents and two US patents.

His research interests include Internet architecture and protocols, media streaming, wireless ad hoc networks, and service overlay networks. He serves as TPC member for various networking conferences, including IEEE INFOCOM, IEEE MASS, and IWQoS. He was TPC Co-Chair for The First IEEE International Workshop on Multimedia Systems and Networking (WMSN05), Information System Co-Chair for IEEE INFOCOM04, and a guest-editor for ACM/Kluwer Journal of Mobile Networks and Applications (MONET), Special Issues on Wireless Sensor Networks and Wireless Mesh Networks. He is an editor of IEEE Communications Surveys and Tutorials. He is a member of IEEE and IEEE Communications Society, and an elected member of Sigma Xi.

Ion Stoica is founder and the Chief Technology Officer of Rinera Networks, Inc., and Associate Professor in the EECS Department at University of California at Berkeley. He has done research on peer-to-peer network technologies in the Internet, resource management, and network architectures. Stoica is the recipient of a Sloan Foundation Fellowship (2003), a Presidential Early Career Award for Scientists & Engineers (PECASE) (2002), and of the ACM doctoral dissertation award (2001).

Hui Zhang is founder and president of Rinera Networks, Inc., and Professor in the School of Computer Science at Carnegie Mellon University. He has done research on clean slate Internet architecture, Internet QoS, multicast, and peer-to-peer video streaming systems.

Zhang is an ACM Fellow. He received the National Science Foundation CAREER Award in 1996 and the Alfred Sloan Fellowship in 2000. He held the CMU SCS Finmeccanica Chair from 1998 to 2002. He was the Chief Technical Officer of Turin Networks in 2000-2003.



Xinyan Zhang (S03) received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 2001 and the M.Phil. degree from the Department of Information Engineering, Chinese University of Hong Kong in 2004. He developed and implemented Coolstreaming, one of the largest Peer-to-Peer global live streaming systems at the time (2004-2005). He is with Roxbeam Corp.