Inside the New Coolstreaming: Principles, Measurements and Performance Implications

Bo Li*[†], Susu Xie[†], Yang Qu[‡], Gabriel Y. Keung[†], Chuang Lin[‡], Jiangchuan Liu[§] and Xinyan Zhang[††]
[†]Hong Kong University of Science and Technology, [‡]Tsinghua University
[§]Simon Fraser University, [††]Roxbeam Inc.

***Abstract*—The Peer-to-Peer (P2P) based video streaming has emerged as a promising solution for Internet video distribution. Leveraging the resource available at end users, this approach poses great potential to scale in the Internet. We have now seen the commercial P2P streaming systems that are orders of magnitude larger than the earlier academic systems. We believe understanding its basic principles and limitations are important in the design of future systems.**

The Coolstreaming, first released in summer 2004, arguably represented the first successful large-scale P2P live streaming system. Since then, the system has been significantly modified and commercially launched. This paper takes an inside look at the new Coolstreaming system by exposing its design options and rationale behind them, and examines their implications on streaming performance. Specifically, by leveraging a large set of traces obtained from recent live event broadcast and extensive simulations, we study the workload characteristics, system dynamics, and impact from a variety of system parameters. We demonstrate that there is a highly skewed resource distribution in such systems and the performance is mostly affected by the system dynamics. In addition, we show that there are inherent correlations and fundamental trade-off among different system parameters, which can be further explored to enhance the system performance.

I. Introduction

Live video streaming is perhaps the greatest unfulfilled promise in the Internet. Earlier proposals have largely built upon the IP multicast framework [1], in which they rely exclusively on routers for construction of dynamic tree(s) for video multicast. While IP multicast is efficient and has attracted significant attention, its deployment remains limited due to many practical issues. Recent development has been focusing on the use of the Peer-to-Peer (P2P) technology [2], which enables quick and easy deployment of multicast functions and eliminates the needs for infrastructure support. Nevertheless, this paradigm results in a new set of challenges that must be addressed, in particular, real-time video delivery and the dynamics of autonomous end-hosts.

In a typical overlay multicast architecture, end systems self-organize themselves into an overlay topology and data forwarding follows links of the overlay. The tree topology is probably the most natural and efficient structure, and most of the earlier proposals have exclusively adopted the use of a tree for overlay multicast [3] [4]. The tree, however, suffers from node dynamics and thus is subject to frequent reconstruction, which incurs extra cost and renders to sub-optimal structure. We designed and implemented a completely different approach in Coolstreaming [5] called *data-driven* overlay. The key novelties were: 1) peers gossip with one another for content availability information, thus it can independently select neighboring node(s) without any prior-structure constraint; 2) the content delivery is based on swarm-like technique using pull operation. This essentially creates a mesh topology among overlay nodes, which is shown to be robust and very effective against node dynamics. Since its release, several other systems such as PPlive [6] and Sopcast [7] have also been successfully launched that have attracted millions of viewers.

Coolstreaming arguably represented the first large-scale P2P video streaming experiment [5] , which has been widely referenced in the community as the benchmark (Google entries once topped 370,000) that a P2P-based live streaming system has the potential to scale. While the system exhibited reasonable video playback quality, there were, however, two main drawbacks in the earlier system: 1) long initial start-up delay due to the random peer selection process and per block pulling operation, and 2) high failure rate in joining a program during flash crowd. Since its first release, while keeping the basic mechanisms intact, we have completely redesigned and implemented the Coolstreaming system [8], specifically: 1) we have now implemented a hybrid pull and push mechanism, in which the video content are pushed by a parent node to a child node except for the first block. This helps to significantly reduce the overhead associated each video block transmission, in particular the delay in retrieving the content; 2) a novel multiple sub-streams scheme is implemented, which enables multi-source and multi-path delivery of the video stream. Observed from the results, this not only enhances the video playback quality but also significantly improves the effectiveness against system dynamics; 3) the buffer management and scheduling schemes are completely re-designed to deal with the dissemination of multiple sub-streams; 4) multiple servers are strategically deployed, which substantially reduce the initial start-up time to under 5 seconds, thus meeting the commercial system requirement.

There are many issues relevant to the design of live video streaming systems such as system dynamics, heterogene-

*The corresponding author: E-mail: bli@cse.ust.hk; Tel: +852 2358 6976; Fax: +852 2358 1477

The research was support in part by grants from RGC under the contracts 616505, 616406 and 616207, by a grant from NSFC/RGC under the contract N_HKUST603/07, by a grant from HKUST under the contract RPC06/07.EG27, by grants from NSF China under the contracts 60429202, by a grant from National Basic Research Program of China (973 Program) under the contract 2006CB303000.

ity, peer churn, network congestion, stream synchronization, multi-path routing, topology stability and convergence [4] [9] [10]. There have been investigations on the optimal resource allocation [11], scaling factors [12], incentive-based mechanism [13], fine-granularity control [14], priority based multicast [15], and integration with network encoding technique [16]. However, there seems to be several inconsistencies in the most basic understanding of a live video streaming system. We believe understanding the fundamental principles, design trade-off and system dynamics are the important steps in the design of future system. Further, there has been little experimental study on large-scale P2P streaming systems. The main constraint in prior studies was the lack of knowledge in the architecture and control mechanisms due to the proprietary nature of the commercial P2P streaming systems, in which few measurement studies [17] [18] could only seek for mechanisms such as packet sniffing in trying to capture the external behaviors of such a system. This, however, can not provide insights into its fundamental system design trade-offs and also is difficult to offer rational explanations for the engineering decisions associated with that.

This paper takes a different approach by contributing to the basic understanding: 1) we describe the basic principles and key components in a real working system; 2) we examine the workload characteristics and the system dynamics; 3) we analyze from real traces what the key factors affect the streaming performance; and 4) we investigate the sensitivity from a variety of system parameters and offer our insights in the design of future systems. Our purpose in this paper is to demonstrate concretely what set of realistic engineering problems exist and how a working system resolves some of these issues. To the best of our knowledge, this is the first paper that closely examines a large-scale commercial P2P streaming system with full knowledge of the internal mechanisms, and further studies the impact from a variety of system parameters.

II. Related works

The existing work on P2P live video streaming generally can be classified into two categories: tree-based overlay multicast and data-driven approaches. In the tree-based approach, the key is to construct a multicast tree among end hosts. The End System Multicast was one of first such protocols [3], which examined the main difficulties encountered in the native IP multicast and proposed moving the multicast functionalities to the edge of the networks. The construction of a multicast tree was done using an overlay network. It demonstrated the feasibility of implementing multicast functions at end systems while keeping the core functionalities of the Internet intact. This single-tree based overlay suffers two drawbacks: 1) potentially poor resource usage and unfair contributions in that a leaf node cannot contribute upload capacity; 2) given the dynamic nature of nodes, the departure or failure of high-level node can cause significant program disruption and requires the re-construction of the overlay topology. This can potentially result in poor performance in the event of churns. The multi-tree approach has been proposed to cope with this problem [19] [20], which usually requires that the media source encodes a video stream into multiple sub-streams and each sub-stream is distributed over a separate overlay tree. This achieves two improvements, 1) better resilience against churn, and 2) fairer bandwidth utilization from all peer nodes. However a multi-tree scheme is more complex to manage in that it demands the use of special multi-rate or/and multi-layer encoding algorithms, which has not been demonstrated feasible in real systems over the Internet. Further, this often requires that multiple trees are disjoint, which can be difficult in the presence of network dynamics [15].

There have been several recent works such as Chunkyspread [14], in which it focused the efficiency associated multi-trees construction and also explored the simplicity and scalability adopted from unstructured networks. Rao et al. evaluated the multi-tree framework and proposed a new concept called contribution awareness as an incentive to enable better contribution from peer nodes [21]. Reza et al. proposed to use swarming technique to enable leaf nodes to contribute in uploading capacity [22]. There were also tree-less proposals [23] [24], which was also adopted in Gridmedia [25].

Within the framework of P2P live streaming, there are several fundamental questions that are unclear such as how an overlay evolves under a random peer selection scheme, under what condition that an overlay can be stabilized, in what sense is the system scalable and what are the fundamental limitations and trade-offs. None of the existing works have provided a comprehensive study on these important issues. In this paper, we demonstrate the real engineering problems we have encountered; we concretely describe how those problems are resolved with various mechanisms, how different components interact with one another, and what are the impact from various system parameters. We believe this is important in the design of future streaming systems.

III. The New Coolstreaming - Redesign and Key Components

Before we describe the architecture and components in the new Coolstreaming system, it is worth pointing out a few facts. Coolstreaming was developed in Python language earlier 2004. Since the first release (Coolstreaming v0.9) in March 2004 and until summer 2005, it had attracted millions of downloads worldwide, and the peak concurrent users reached 80,000 with an average bit rate of 400Kbps. The system became the base technology for Roxbeam Inc., which has launched the commercial streaming services in multiple countries.

A. Basic Components

There are two basic functionalities that a P2P streaming system must have: 1) from which node one obtains the video content; and 2) how the video stream is transmitted. The Coolstreaming system adopted a similar technique initially used in BitTorrent (BT) for content location, i.e., using a random peer selection; it then uses a hybrid pull and push mechanism in the new system for content delivery. One of the central designs 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. This offers a few 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.

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 partnership with other peers and also exchanges the availability of video content using *Buffer Map* (BM) with peer nodes. 3) *Stream manager*, which is responsible for data delivery.

In the Coolstreaming system, 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 *bootstrap node* for a list of nodes and stores that in its own mCache. It then selects a few nodes randomly to initiate the content retrieval. There is an important distinction between *parent-children relationship* and *partnership*. The partnership is established between two peer nodes so that they exchange block availability information. The parent-children relationship is established when a node (the child) is actually receiving video content from another node (the parent). Apparently, a node's parents or children are a subset of the nodes from its partnership set.

These basic modules form the base for the initial Coolstreaming system, which has been verified to be effective in the Internet. The new Coolstreaming system [8], however, has made significant changes in the design of other components, which will be described in the rest of this Section.

B. Multiple Sub-Streams

The video stream is divided into *blocks* with equal size, in which each block is assigned a sequence number to represent its playback order in the stream. One of the major difficulties in video streaming application is to maintain the continuity of the video stream in the presence of network dynamics. We divide each video stream into multiple sub-streams without any coding, in which each node can retrieve any sub-stream independently from different parent nodes. This subsequently reduces the impact due to a parent departure or failure, and further helps to achieve better transmission efficiency similar to the P2P file transfer protocols [26]. The complication is the synchronization among different sub-streams, which has to be carefully maintained.

In general, a video stream is decomposed into K sub-streams by grouping video blocks according to the following scheme: the i-th sub-stream contains blocks with sequence numbers $(nK+i)$, where n is a non-negative integer, and i is a positive integer from 1 to K. This implies that a node can at most receive sub-streams from K parent nodes. An example is given in Fig. 1, in which K equals 4.

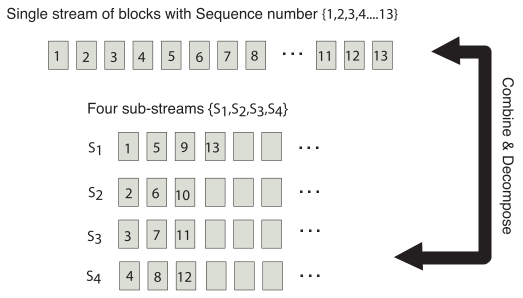
Single stream of blocks with Sequence number {1,2,3,4....13}
1 2 3 4 5 6 7 8 ... 11 12 13
Four sub-streams {S1,S2,S3,S4}
S1 1 5 9 13
S2 2 6 10
S3 3 7 11
S4 4 8 12
Combine & Decompose

Fig. 1. An example of stream decomposition

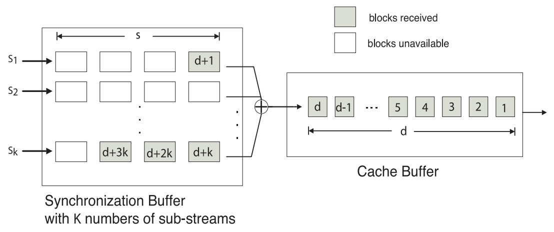

Fig. 2. Structure of buffers in a node

C. Buffer Partitioning

Buffer Map or BM is introduced to represent the availability of latest blocks of different sub-streams in buffer. This information also has to be exchanged periodically among partners in order to determine which sub-stream to subscribe to. The Buffer Map is represented by two vectors, each with K elements. The first vector records the sequence number of the latest received block from each sub-stream. The sub-streams are specified by $\{S_1, S_2, ..., S_K\}$ and the corresponding sequence number of the latest received block is given by $\{H_{S_1}, H_{S_2}, ..., H_{S_K}\}$. For example, $\{2K+1, 3K+2, 2K+3, ..., 4K\}$ implies that a node receives blocks up to sequence number "$2K+1$" from the first sub-stream, receives blocks up to sequence number "$3K+2$" from the second sub-stream and so on. The second vector specifies the subscription of sub-streams from the partner. For example, if a node A is requesting the subscription of the first and second sub-streams from node B, it sends out a message $\{1, 1, 0, 0, ..., 0\}$ to the node B. In summary, BM specifies the block availability information and the current requests. By exchanging the BM among partners, a node can request subscriptions for particular sub-stream(s) from its partner by a single request.

In the new Coolstreaming, each node maintains an internal buffer, whose structure is illustrated in Fig. 2. 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. The blocks will be stored in chronological order in the synchronization buffer according to their sequence numbers. They will be combined into one stream when blocks with continuous sequence numbers have been received from each sub-stream.

D. Push-Pull Content Delivering

In the original Coolstreaming system, each block has to be pulled by a peer node, which incurs at least one round delay per block. The new Coolstreaming adopts a hybrid *push and pull* scheme. When a node subscribes to a sub-stream by connecting to one of its partners via a single request (pull) in BM, the requested partner, i.e., the parent node, will continue pushing all blocks in need of the sub-stream to the requested node. This not only reduces the overhead associated with each video block transfer, but more importantly, significantly reduces the timing involved in retrieving video content.

With the exchange of BM information, a newly joined node can obtain the availability of blocks from its parent nodes. One crucial question is from which part of the stream that the newly joined node should request the video content. This turns out to be a non-trivial problem. Suppose the range of the blocks available 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 that, if the node requests the block starting from the highest sequence number m, the partner nodes might not have sufficient subsequent blocks to maintain stream continuity; 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 after being played back; 2) it incurs long initial start-up delay for the newly joined node to catch up with the current video stream. We define a system parameter T_p to determine the initial block number, which will be specified next. Once the initial sequence number is determined, the node checks the availability of blocks in its partners' BMs and then selects appropriate partner nodes as its parents for each sub-stream. Noticing that this process is essentially the same as when a node needs is to re-select a new parent due to the parent node departure or failure. After that, parent nodes can then continue pushing the available blocks of sub-streams to the newly joined node.

E. Parent re-selection

We next describe the process how a node re-selects a new parent in case of churn.[*] Each peer node needs to constantly monitor the status of the on-going sub-stream transmissions and re-selects new parents when existing connections are inadequate in satisfying the streaming requirement. The question is what triggers the parent re-selection.

To address this problem, we define two metrics to specify the uploading capacity: $\{T_s, T_p\}$, which are 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 at node A. T_p is defined as the threshold of the maximum sequence number deviation of the latest received blocks between the partners of node A and the parents of node A. We denote $H_{S_i,A}$ as the sequence number of 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 are introduced:

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

$$\max\{H_{S_i,q} : i \le K, q \in parnters\} - H_{S_j,p} < T_p \qquad (2)$$

The Inequality (1) is used to monitor the buffer status in the received sub-streams of 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 insufficient upload capacity for this sub-stream, which will subsequently trigger the peer re-selection. 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 switch to a new parent node from its partners.

IV. Log & Data Collection Results

In this Section, we present and study the results obtained from a representative live event broadcast on 27th September, 2006 in Japan. A sport channel had a live baseball game that was broadcasted at 17:30 and we recorded real traces from 00:00 to 23:59 on that particular day. Specifically, we focus on the two important metrics in this paper due to the length limitation; one is the uploading capacity distribution in the system and the other is the key factor that affects the overall performance.

A. System Configuration

There is a log server in the system. Each user reports its activities to the log server including events and internal status periodically. Users and the log server communicate with each other using the HTTP protocol. There are three types of reports for each peer node: 1) video quality report, 2) traffic report such as the amount of uploading and downloading and 3) activity report such partner updates.

Each video program is streamed at a bit rate of 768 Kbps. To provide better streaming service, the system deploys a number of servers (24 in total), as shown in Fig. 3. The source sends video streams to the servers, which are collectively responsible for streaming the video to peers. In other words, users do not directly retrieve the video from the source. This improves the overall performance in two ways: 1) the streaming capacity is seemingly amplified proportionally with the number of servers; and 2) the content is pushed closer to the users.

B. User Types and Distribution

Upon initiating a join operation, each user is assigned a unique ID. The log system also records the IP address and port number for the user. The *total number* of users represents the number of users (each having a unique ID) in the system at any instance of time.

We classify users based on the IP address and TCP connections into the following four types: 1) *Direct-connect*:

[*]The procedure for a new joined node to select parent nodes is similar and even simpler, so we do not describe the peer joining process in this paper

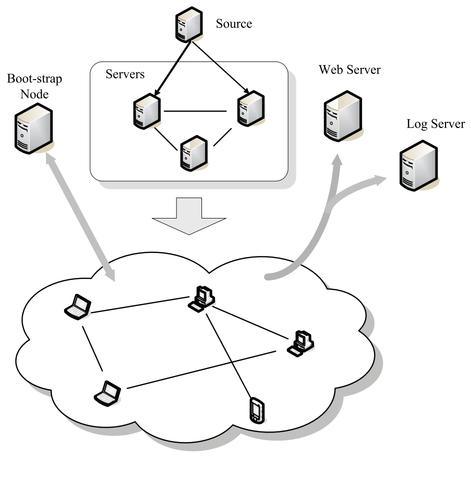

Fig. 3. Coolstreaming System Configuration

peers have public addresses with both incoming and outgoing partners; 2) *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; 3) *NAT*: peers have private addresses with only outgoing partners; 4) *Firewall*: peers have public addresses with only outgoing partners.

Fig. 4a plots the number of sessions in the system in a typical day, and Fig. 4b plots the number of sessions in the system between 18:00-24:00. These illustrate user behavior in a typical weekday and during the peak hours respectively. The sudden drop in the number of users around 22:00 is caused by the ending of some programs. This shows that the system can scale in that the number of nodes in the system can quickly ramp up to $40,000$.

Fig. 5a and Fig. 5b plot the different types of users in the system for the whole day and between 18:00-24:00. Observed from the figures, we can see that 1) 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 [5] ; 2) there are a significant number of NAT users (more than 45%) with limited uploading capabilities, since the current system did not use any NAT traversal technique; 3) there are some users that we cannot determine their connection types in time. This is primarily caused by either the delay in log report when the log server is busy or partnership is not stabilized before we can determine the connection type.

C. System Dynamics

We next examine performance impact from system dynamics; specifically, we focus on the failure rate in the system. This is because the critical performance impact on the users that we have observed is during the initial joining phase, in which there could be users failing to start the program. This occurs when a user fails to start playing the video within a time limit (usually 20-30 seconds) and the system automatically triggers the reconnection to a new set of peer nodes. This is caused by the fact that one or more connected peer is incapable of providing the sub-stream(s) due to overloading, networking congestion, NAT, or other reasons. Fig. 6a and 6b plot the correlation between join rate, leave rate, and failure rate. This clearly demonstrates that there exists a strong correlation between failure events and join events, as well as between failure events and leave events. We further plot the failure rates against the number of users in the system in Fig. 6c, which does not show any noticeable correlation between them. This implies that the failure is mostly caused by churn, and not affected by the size of the system.

The rationale behind the above observations is how the overlay is constructed and maintained in the new Coolstreaming. Specifically, each peer maintains a partial view of the overlay through the mCache. The update of the mCache entries is achieved by randomly replicating entries when new partnership is established. This results that older peers or less active peers will be faded out from mCache gradually. However, during flash crowds, mCache may be filled with too many newly joined peers, which often cannot provide stable video streams. Under such a replication algorithm, it thus takes longer time for a newly joined peer to obtain video stream.

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. If the aggregate upload capacity from a user is zero, the contribution index is also zero. This implies that the user does not contribute any uploading capacity in the system. 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, which indicates that the user is capable of providing full video streams to another user. In order to better capture the granularity of the uploading contribution from each user, we categorize user contributions into levels by their average values of the contribution index: 1) *Level 0*: contribution index is larger than one. The user can upload the whole video content to another user; 2) *Level 1*: contribution index is between 0.5 and 1. The user can at least upload half video content to its children; 3) *Level 2*: contribution index is between 1/6 and 0.5. The user can upload at least one sub-stream to its children; and 4) *Level 3*: contribution index is less than 1/6. The user cannot stably upload a single sub-stream to its children.

Fig. 7a plots the average contribution index distribution from all users during the broadcast. This shows that significantly number of users (65%) cannot stably contribute one sub-stream. We believe this is caused by the fact most of the users are NAT and firewalls (see Fig. 5a). Fig. 7b plots the contribution index for different types of users against time. The fluctuation is caused by the fact that there are users whose connection types cannot be determined. The results show a strong correlation between the contribution index and the user types. For example, it is conclusively shown that the

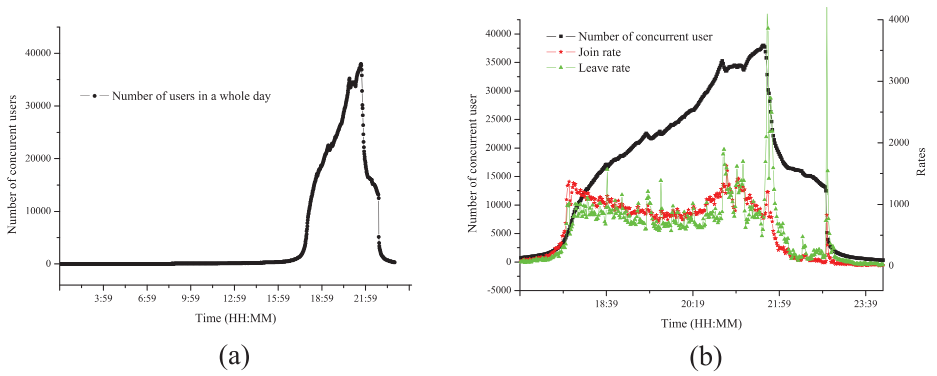

Fig. 4. (a) The evolution of the number of users in the system in a whole day; (b) The evolution of the number of users in the system from 18:00 to 23:59

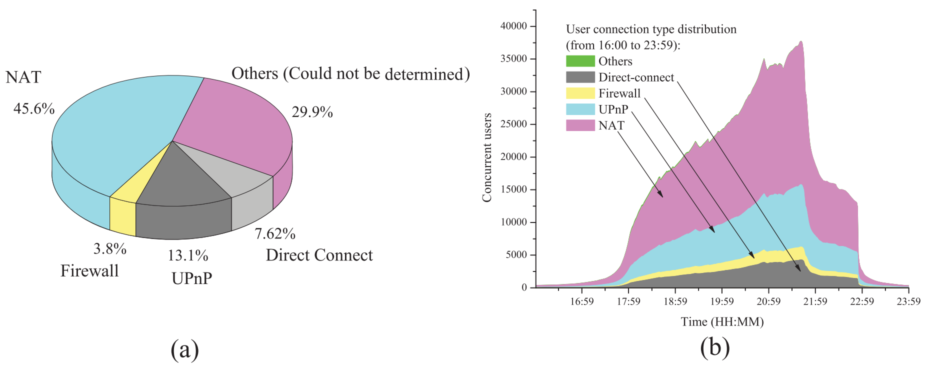

Fig. 5. (a) User types in a whole day; (b) User types from 18:00 to 23:59

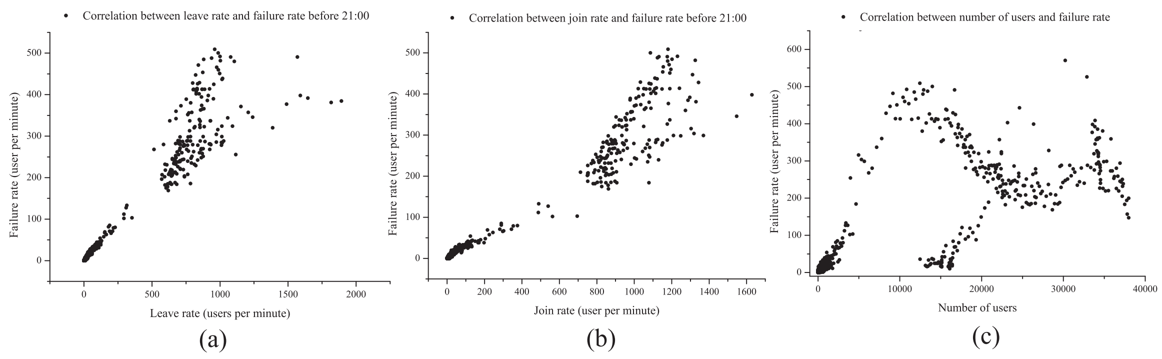

Fig. 6. (a) The correlations between join rate and failure rate; (b) The correlations between leave rate and failure rates; (c) Correlation between failure rate and system size

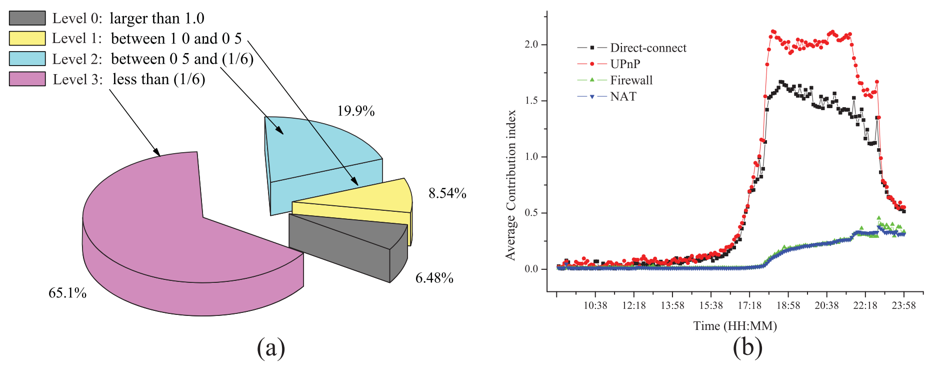

Fig. 7. (a) The distribution of contribution index from Level 0 to Level 3; (b) Average contribution index of different user connection types against time

contribution index increases with the size of the system, and UPnP and directly connected users contribute far more than users behind firewalls or NAT.

V. Sensitivity Analysis

The results in the previous section clearly demonstrate that there are strong correlations among different components and the system performance is affected by various parameters. However, it is not feasible to derive the performance under different settings from live event broadcast. In the section, we build simulation and analyze the correlations and sensitivity.

A. Simulation Setting

We build a simulation that captures all detailed functionalities in the new Coolstreaming, including overlay construction, partnership maintenance, BM management and content delivery. We make a few simplifications in the simulation based on realistic settings: 1) since the capacity inside the network has been found to be generally sufficient enough ($\geq$ 5 Mbps) [27], the edge is likely to be the bottleneck in the system. Further, the downloading capacity is usually much larger than the uploading capacity; we assume the system bottleneck belongs to the out-going (uploading) capacity; 2) the simulation focuses on the block-level performance and does not model the packet-level dynamics of TCP connection; 3) we ignore the signaling overhead caused by the exchange of the BM since only one-bit is required for each video block in the BM.

The delay needs to take into account the number of connections that are sharing the uploading capacity, which is varying with time. For the network propagation delay, the average value is 79 milliseconds according to the real-world node-to-node latency matrix (2500×2500) measured in the Internet [28]. We model the network propagation delay by a random value within $[5, 155]$ milliseconds, which is believed to be appropriate in general cases.

The following are the default parameters in the simulation. The video stream is coded with 400Kbps with 10 sub-streams and each block is 10K bits. There is one source node and one bootstrap node in the system initially. At the beginning of the simulation, 5,000 nodes join the system according to *Poisson arrival* with an average inter-arrival time 10 milliseconds. The source node has upload capacity of 9Mbps and can handle up to 15 children (partners). The bootstrap node can maintain a record of 500 nodes and its entry can be updated. Each node can maintain partnerships with at most 20 peers and can buffer up to 30 seconds' content. Each node can start playing the video when the buffer is half-loaded. We consider both a homogeneous setting in which each node has an equivalent uploading capacity (500Kbps), and a highly heterogeneous setting in which nodes have uploading capacity at 100Kbps and 900Kbps.

We use the following metrics to evaluate the system performance. (i) Playback continuity (continuity index): the ratio of the number of blocks that exist in buffer at their playback due time to that of blocks that should have been played over time, which is the main metric for evaluating user satisfaction. (ii) Out-going (Uploading) bandwidth utilization: a metric for evaluating how efficient the uploading bandwidth capacity is used. (iii) Effective data ratio: the percentage of useful data blocks in all the received ones. This metric evaluates how efficient the bandwidth is utilized. (iv) Buffer utilization: this measures how the buffer is utilized. (v) Startup delay: the waiting time until a node starts playing the video after it joins the system. (vi) Path length: the distance between a node and the source in the overlay. This metric characterizes the overlay structure. (vii) Partner/parent/child change rate(s): The changes in partners/parents/children per second within a node, which is a metric for evaluating overlay stability.

B. Sensitivity Analysis

We are primarily concerned with the impact on the streaming performance from main system parameters including block size, number of sub-streams, the number of partners and upload capacity.

Fig. 8-12 illustrate how the continuity index and upload capacity utilization behave with under different block size, number of partners, number of sub-streams, initial buffering ratio and join rate. In Fig. 8, the block size varies from 10K bits to 100K bits. It is shown that the continuity index, upload capacity utilization, and effective data ratio all decline with the increase of block size. The decrease of effective data ratio is due to the data redundancy caused by larger block size, in which a node can possibly retrieve a portion of same block from its new parent after parent re-selection. The decrease of upload capacity utilization due to larger block size is caused by two reasons: 1) longer transmission time for each block, and 2) less frequent update of the block information in the buffer map, so making it more difficult in finding a parent node in time. Consequently, the continuity index is affected by both lower upload capacity utilization and effective data ratio. It is evident that larger number of partners increases the probability in finding capable parent and multiple sub-streams can reduce the latency and the impact from parent failure or departure. These are verified in Fig. 9 and Fig. 10. In addition, Fig. 10 also shows that the increase of the number of sub-streams beyond 8 does not bring any further improvement due to the complication in handling multiples sub-streams in each node.

In Fig. 11, the initial buffering ratio represents how much video content has been accumulated in buffer before playback, which varies from 0.3 to 0.9. It is shown that the continuity index, upload capacity utilization increase with the increase of initial buffering ratio. This is anticipated as more content is available at each node for sharing with its children nodes. The trade-off, however, is the potentially longer delay. In Fig. 12, the join rate varies from 30 to 100 nodes per second. It is shown that the continuity index is not sensitive to join rate, which implies that the system is effective against different join rates. However, with a higher join rate, the average uploading contribution will be effectively reduced since the newly joined nodes do not have sufficient content for uploading to others.

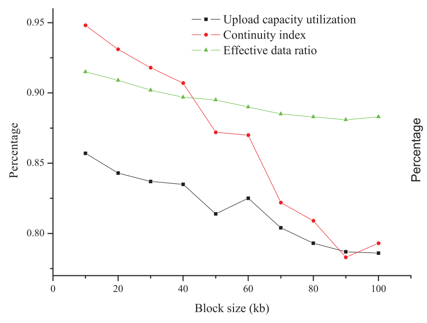

Fig. 8. Continuity index and upload utilization against block size

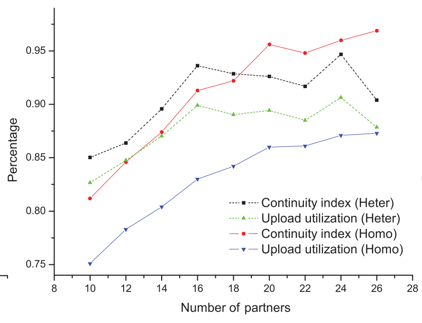

Fig. 9. Continuity index and upload utilization against number of partners

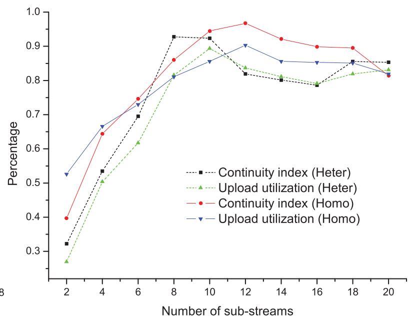

Fig. 10. Continuity index and upload utilization against number of sub-streams

We next investigate the fundamental trade-offs in system performance under different system parameters. Fig. 13 demonstrates the trade-off between the continuity index and startup time under different number of sub-streams. As shown in Fig. 10, multiple sub-streams can improve the continuity index, but this does not come without a price. It can be observed from Fig. 13 that the startup time generally increases with the increase of the number of sub-streams except for the case when this is changed from 2 to 4. The increase of the startup time is mainly caused by the variations and synchronization in receiving sub-streams from different parents.

Fig. 14 illustrates how the path length evolves over time under different number of sub-streams. As defined earlier, the path length characterizes the overlay stability. The results demonstrate that 1) the overlay can all gradually converge to a stable state; and 2) using a larger number of sub-streams, while improving other performance index in particular the continuity index, can result in slower overlay convergence. This is caused by the fact that a node needs longer time to retrieve a larger number of sub-streams from different parents. We next analyze how system performance evolves over time. Fig. 15 and 16 illustrate how partner/parents/children change under different upload capacity; specifically the upload capacity varies from 425Kbps to 500Kbps by a step of 25Kbps. Fig. 15 shows that the partner change rate decreases sharply at the beginning, and then quickly converges to a stable rate. The continuity indexes obtained are 0.78 when the uploading capacity is 425Kbps and 0.94 when the uploading capacity is 500Kbps. This indicates that the overlay can reach a stable state within 2 minutes after the initial node joining (flash crowd phase), and higher uploading capacity can lead to more stable topology (i.e., less partner change) and better video playback quality. Fig. 16 shows that the parent/children change rates also converge to stable rates under different settings within 2-3 minutes, and the stability is sensitive to the uploading capacity.

VI. Conclusions

In this paper, by leveraging a set of real traces obtained from the recent live event and extensive simulation, we explored the design in the new Coolstreaming and closely examined its performance implications. We studied the workload, inherent system dynamics, and performance measurement, and 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 a P2P streaming system is the excessive start-up time and high join failure rates during flash crowd; 2) the system dynamics 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; and 4) there exists critical system design trade-off in various system parameter settings.

We believe the lengthy start-up time and high failure rates are inherent problems in P2P streaming systems, especially in the presence of large percentages of the nodes behind NAT/firewalls. Further, when there are fewer users in the system, for example, at the beginning of a program, finding a capable partner often takes longer time. Thus we believe that the deployment of a certain number of server is of necessity in a live streaming system. On the other hand, pure server-based approaches like in CDN can be costly and do not scale well. P2P provides excellent complementary solution due to its self-scaling property. Therefore, we believe that a large-scale commercial Internet streaming system should be a hybrid system.

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. First, the data set does not allow us to derive accurate overlay topology and pair-wise performance. We believe it is of great relevance to examine how the overlay topology migrates and self-stabilizes under the random topology construction. Second, it would be of interest to evaluate the actual resource distribution in the system and how it impacts on the video playback quality. Third, there are other optimizations that could be explored to improve the system performance, such as clustering geographical vicinity nodes and incorporating the nodes and traffic behavior in server placement.

References

[1] S. Deering, “Multicast Routing in Internetworks and Extended LANs,” in *Proc. of ACM SIGCOMM*, August 1988.
[2] J. Liu, S. Rao, B. Li and H. Zhang, “Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast,” (invited) *Proceedings of the IEEE*, *Special Issue on Recent Advances in Distributed Multimedia Communications*, 2007.

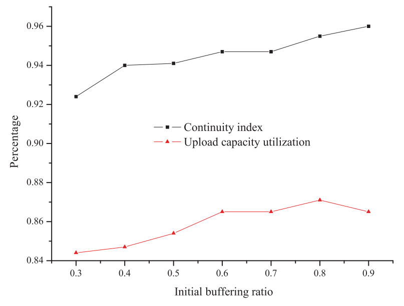

Fig. 11. Continuity index and upload utilization against initial buffering ratio

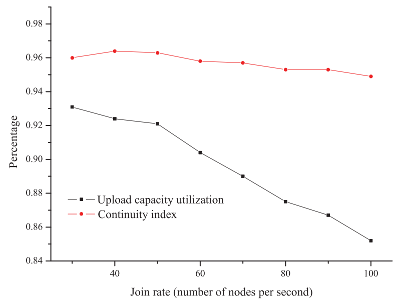

Fig. 12. Continuity index and upload utilization against join rate

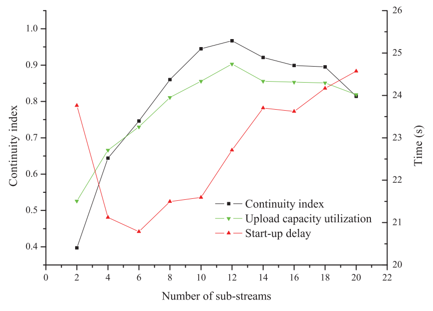

Fig. 13. Trade-off comparison between continuity index and start-up delay (different number of sub-streams)

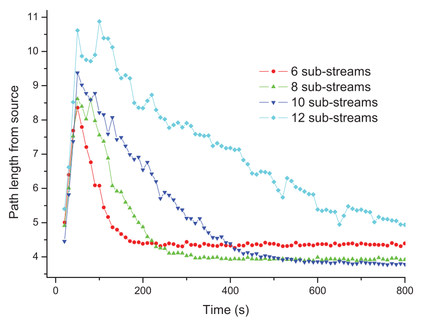

Fig. 14. Path length against simulation time under different number of sub-streams

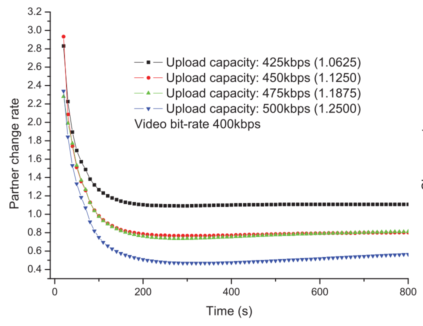

Fig. 15. Partner change rate against time under different upload capacity

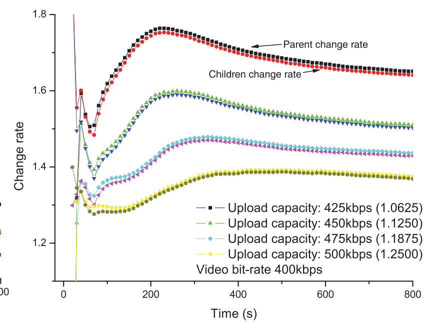

Fig. 16. Parent/children change rate against time under different upload capacity

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

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. of ACM SIGCOMM*, August 2002.

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

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

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

[8] Susu Xie, Bo Li, Gabriel Y. Keung, and Xinyan Zhang, Coolstreaming: Design, Theory and Practice," in *IEEE Transactions on Multimedia*, Vol. 9, Issue 8, December 2007.

[9] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient Multicast Using Overlays," in *Proc. of ACM SIGMETRICS*, June 2003.

[10] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," in *IEEE Journal on Selected Areas in Communications*, Vol. 20 No. 8, October 2002.

[11] Y. Cui, Y. Xue and K. Nahrstedt, "Optimal Resource Allocation in Overlay Multicast," in *Proc. of IEEE ICNP*, November 2003.

[12] T. Small, B. Liang, and B. Li, "Scaling Laws and Tradeoffs in Peer-to-Peer Live Multimedia Streaming", in *Proc. of ACM Multimedia'06*, October 2006.

[13] W. Wang and B. Li, "Market-based Self-Optimization for Autonomic Service Overlay Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 23, No 12, December 2005.

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

[15] M. Bishop, S. Rao, and K. Sripanidkulchai, "Considering Priority in Overlay Multicast Protocols under Heterogeneous Environments," in *Proc. of IEEE Infocom*, April 2006.

[16] M. Wang and B. Li, "Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming," *Proc. of IEEE Infocom*, May 2007.

[17] 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 Services over World Wide Web in conjunction with WWW2006*, May 2006.

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

[19] 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.

[20] 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.

[21] Y. Sun, M. Bishop and S. Rao, "Enabling Contribution Awareness in an Overlay Broadcast System," in *Proc. of ACM SIGCOMM*, September 2006.

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

[23] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination using an Overlay Mesh," in *Proc. of ACM SOSP*, October 2003.

[24] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast," in *Proc. 4th International Workshop on Peer-to-Peer Systems*, February 2005.

[25] 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.

[26] 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.

[27] A. Akella, S. Seshan, and A. Shaikh, "An Empirical Evaluation of Wide-Area Internet Bottlenecks," in *IMC*, 2003.

[28] http://www.cs.cornell.edu/people/egs/meridian/data.php