

On the Design of Incentive-Aware P2P Streaming

Tianhao Qiu, Ioanis Nikolaidis, and Fulu Li

Abstract—We study peer-to-peer (P2P) multicast streaming protocols for high bandwidth streams to large numbers of heterogeneous and transient users. First, the shortcomings of tree-based P2P schemes are discussed. Subsequently, a scalable swarm-based scheme is introduced that avoids maintaining a rigid logical topology. Peers self-organize into an unstructured overlay in an ad-hoc fashion. Stream data are divided into small-sized units and distributed in a fashion that fully utilizes the upload capacity of all peers. A credit-based incentive scheme is introduced to encourage peers to contribute their capacity. The incentive scheme operates based exclusively on local information. We evaluate the proposed scheme through simulations in a dynamic and heterogeneous environment. A key benefit of the proposed scheme is its ability to operate under resource-constrained conditions where traditional tree-based approaches typically fail.

Index Terms—peer-to-peer networks, streaming protocols, incentive schemes.

I. INTRODUCTION

Live content streaming is becoming increasingly popular among users with broadband access. Setting up a unicast connection from server to each served client is an inefficient solution. Technically speaking, multicast is the best known approach to support scalable live media streaming. However, the deployment of multicast has been slow for a variety of reasons [9]. Application Level Multicast (ALM) has been proposed as an alternative to IP multicast. ALM builds an overlay network via unicast connections between end hosts. A number of P2P live streaming systems based on ALM have been proposed [3], [5], [6], [8] and they share the common trait of an underlying tree (or tree-like) logical topology.

In this paper, we first identify two inherent problems of tree-based overlay multicast: (a) resource underutilization, and (b) premature saturation. We argue that premature saturation undermines the sustainability of a tree under constrained bandwidth conditions. To this end we propose a P2P live streaming system based on an unstructured swarm overlay, where each node is connected to a random subset of peers. To fully utilize upload capacities smaller than the stream bitrate, the stream data are divided into small data unit fragments. Peers transact on the basis of concurrent uploads and downloads of unit fragments. In addition, an “informed” push-based scheduling strategy is introduced which helps propagate data efficiently while reducing duplicate transfers.

A key component of the proposed scheme is a credit-based incentive mechanism that accommodates a certain degree of “unfairness” naturally arising in heterogeneous environments. Namely, rather than ensuring everyone contributes the same amount of bandwidth, and turning away resource-constrained users, it encourages resource-rich nodes to donate more uplink

bandwidth to subsidize resource-constrained peers. Simulation results show that our incentive scheme can motivate capable nodes to upload more because it provides them with better stream quality as well as a higher probability of being placed closer to the source. To our knowledge, this is the first time that incentives via corresponding performance differentiation have been incorporated in a P2P streaming system.

The rest of the paper is organized as follows: In Section II we critically discuss the traditional overlay multicast system based on a single-tree architecture. We detail the design of a swarm-based P2P live streaming system in Section III. We describe our experimental setup and report simulation-based performance results in Section IV. We discuss some related work in Section V and conclude the paper with Section VI.

II. THE CASE AGAINST TREE-BASED SCHEMES

Many proposed P2P live streaming systems (ESM [6], SpreadIt [8]) create and maintain an efficient tree overlay, mimicking a multicast tree structure. However, a tree is inherently vulnerable to network dynamics due to its rigid structure. A data loss or bandwidth fluctuation at a node near the root may affect a large population of downstream nodes. Unlike IP multicast, non-leaf (interior) nodes are autonomous hosts that can fail or may depart the tree at will. When an interior node departs, its (possibly many) descendants suffer stream discontinuity until they find new parents. When connected to the tree, the bandwidth available to any node is limited by the bandwidth available to that node’s parent node. In addition, *only* interior nodes are burdened with forwarding traffic, while leaf nodes do not upload at all. The resulting unfairness is a major disincentive for interior nodes to contribute their bandwidth. Finally, if a peer’s upload capacity is less than the stream bitrate, it can only join as a leaf node. Hence, for high bandwidth streams, only a small percentage of peers can forward the stream at full rate, and the tree can easily become saturated.

In order to quantify the service capacity of the system, we use the *Resource Index (RI)* metric, defined in [28] as the ratio of aggregate upload bandwidth supply (from source and peers) to the total demand for bandwidth in the system ($\#peers \times stream\ bitrate$). *RI* captures the available resources to participating peers. However, in a single-tree structure, the amount of bandwidth that can be utilized depends on the exact capacity of hosts and stream bitrate. For example, given a stream of $512kbps$, a host with an upload capacity of $768kbps$, can support at most one child, leaving $256kbps$ unusable upload capacity. To capture the impact of the unused residual capacity, we define, specifically for trees, RI_{tree} by only considering bandwidth that *can* be used for the construction of the tree given a particular stream bitrate. Clearly, $RI_{tree} < RI$ and its exact value depends on the distribution of upload capacity across peers.

Without loss of generality, for the rest of this paper, we divide hosts into two categories based on their upload capacity: *resource-rich* and *resource-poor*. The former (including the

Manuscript received December 12, 2006; revised June 5, 2007.

This work has been partly supported by a Discovery Grant from NSERC. Tianhao Qiu and Ioanis Nikolaidis are with the Computing Science Department, University of Alberta, Edmonton, AB T6G 2E8, Canada. (e-mails: tianhao.qiu@gmail.com, yannis@cs.ualberta.ca). Fulu Li is with The Media Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA (e-mail: fulu@mit.edu).

source) contribute an amount of upload bandwidth exceeding the stream bitrate, and are able to support at most d children. Resource-poor hosts are assigned an upload capacity less than the stream bitrate. Hence, the resource index $RI_{tree} = d \times \alpha$, where α is the fraction of resource-rich hosts.

RI_{tree} determines a tree's scalability and stability. When $RI_{tree} \gg 1.0$ there are plenty of idle resources available in the system to support newcomers as well as to cope with departing internal nodes by quickly reconnecting its descendants to new parents. When RI_{tree} is closer to (but still more than) 1.0, the system becomes resource-constrained, and the tree grows deeper to accommodate the same number of nodes. The average recovery time for a disconnected node also increases. If $RI_{tree} < 1.0$, the system resources are insufficient to support the current number of participants. New join requests are blocked while descendants of a departing node may be unable to reconnect to the already saturated tree.

A favorable tree-overlay should be balanced to minimize the path length from the root to leaf nodes because longer paths are more prone to failure and congestion. In a balanced tree, a node departure affects on average a smaller number of descendants. Unfortunately, it is not easy to maintain an efficient tree structure due to the existence of resource-poor hosts. Fig. 1 shows a well-balanced tree, where shaded circles represent resource-rich hosts (note $d = 2$ and $RI_{tree} > 1.0$). Fig. 2 shows an unbalanced tree structure with the same set of hosts. We notice the tree becomes unbalanced in Fig. 2 because there are resource-poor peers (6 & 8) connected near the root of the tree. As a result, other resource-rich hosts near the root (node 1 and 2) become critical points in the tree as their descendants consist of most of the population. For example, when node 1 departs, 8 nodes are disconnected in Fig. 2, but only 4 are affected in Fig. 1.

Resource-poor peers near the root also cause *premature saturation* in a dynamic environment. Premature saturation occurs when some critical resource-rich host near the source fails and its position is taken over by a resource-poor host. Theoretically, when $RI_{tree} > 1.0$ all participants should be able to join the tree. However, it is possible for the tree to become saturated prematurely, leaving a large portion of nodes including some resource-rich hosts disconnected. An example of premature saturation is shown in Fig. 3, which could result following a link failure between node 1 and node 2 in Fig. 2. Specifically, Fig. 3 depicts a scenario whereby, after the link failure between nodes 1 and 2, node 7 (a resource-poor host) quickly took over the position of node 2 (a resource-rich host). As a result, node 7 cannot "accept" other downstream nodes due to its resource limitations. Premature saturation is damaging to tree-based schemes. It occurs more frequently in a resource-constrained environment (when resource-poor hosts dominate) which is also when the tree is more unbalanced.

Preemption solves premature saturation by disconnecting resource-poor hosts and replacing them by incoming resource-rich hosts [28]. An example of preemption is shown in Fig. 4. Resource-poor peers are eventually pushed further away from the source, resulting in a more balanced tree. Preemption can also serve as an incentive scheme to reward contributing hosts [22]. The main problem with preemption is its inability to prevent cheating. Users can deliberately misreport information if there is an incentive to do so [25]. If a node declares it possesses more resources than it does and attempts to accept more children than its capacity allows, it impacts its children's performance as well as its own. There

exists no scalable solution to this problem. We may try to measure and verify reported information from peers, requiring complicated techniques to automatically estimate the outgoing bandwidth of peers, but cannot ensure that a peer actually contributes its promised "donation" without testimonials of other peers. A peer can cheat by accepting children but sending little data to them. Isolating cheaters requires children to audit their parent's and complain when "mistreated". But this leaves the possibility of fake complaints and collusion. Overall, preemption is a powerful technique but cannot be applied without careful consideration of its security implications.

III. A SWARM-BASED P2P LIVE STREAMING SYSTEM

P2P swarming or "file swarming" has recently emerged as a popular P2P content distribution mechanism, in particular as implemented by the BitTorrent protocol [7]. Variations have also been proposed [26]. P2P swarming systems do not maintain a structured overlay. The overlay is self-organized in an ad-hoc fashion. Each node is connected to a random subset of peers. Swarm topologies resemble random graphs, and are robust against partition even in the presence of high rate of churn [24], [33]. It has been demonstrated [30] that P2P swarming content distribution is far more efficient than traditional client-server and CDN approaches. Certain unique features of P2P swarming make it also a better match for live streaming compared to tree-based schemes. Specifically:

- A peer in a swarm downloads from multiple suppliers in parallel, enabling it to cope with bandwidth fluctuations and recover from loss of supplier(s) with less quality degradation.
- Data transfers in a swarm are bidirectional, allowing implementation of low overhead and robust incentive schemes, as demonstrated by the tit-for-tat variant used in BitTorrent [7].
- Resource-poor nodes that would have been unable to support children in a tree (little uplink bandwidth) can participate in a swarm by supplying fractions of the stream data.

The mechanism used to implement the last point is the fragmentation of the data stream in small units/granules. We expect that in cases where resources are insufficient, especially when $RI > 1 > RI_{tree}$, a swarm will be able to sustain the demands of all participants while a tree cannot. However, extending P2P swarming from file downloading to live streaming introduces new challenges:

- 1) File swarming achieves high throughput partly because data units are collected out of order, usually following a rarest-first policy to improve data diversity across peers. Live streaming requires data to be delivered (approximately) in order. If peers were to acquire data in strict streaming order, the overall performance of the swarm would deteriorate due to lack of diversity of data/segment sets available across different peers.
- 2) File swarming is tolerant to insufficiency of resources. A lower RI prolongs the waiting time in file swarming, but can seriously degrade playback quality in streaming. If peers find the stream not watchable, they may decide to depart the system.
- 3) Only the average download speed matters in file swarming. In streaming, it is also important to receive data at a relatively constant rate because the playback quality is subject to oscillations in download speed. To ensure the

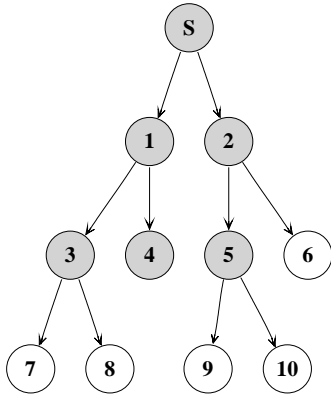


Fig. 1. A well-balanced tree.

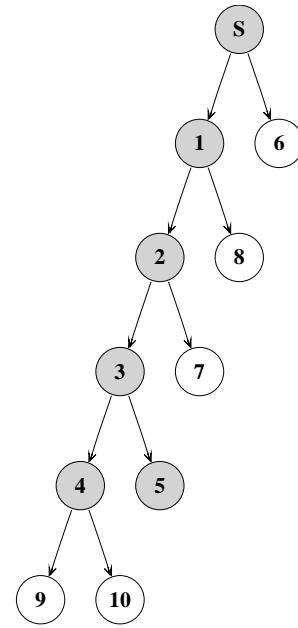


Fig. 2. An unbalanced tree.

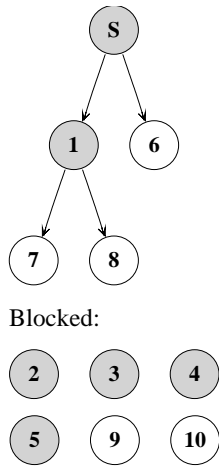


Fig. 3. Premature saturation.

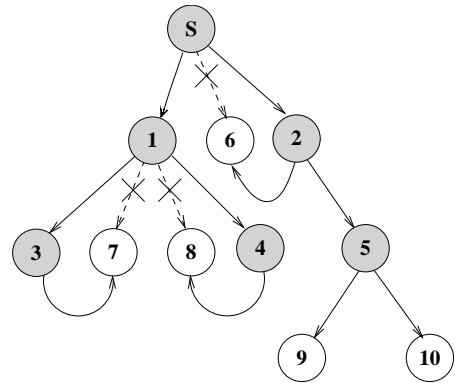


Fig. 4. The result of preemption.

“liveliness” of the playback (i.e., to keep it close to the “live” transmission instants at the source) the delay to disseminate a unit to all the peers must be kept relatively small.

- 4) In file swarming, after the source has distributed a full copy of the file, it functions as a normal peer, either departing the system or becoming a “seed” for future transfers. In live streaming, the source is the only “seed”. If the source cannot distribute enough copies of some data units, those units may not be able to traverse the entire overlay in time for playout.

Next, we present specific design details for a swarm-based P2P live streaming system in detail. Operations that are common in generic file swarming are described concisely.

A. Membership Management

A peer joins by connecting to a number of peers that are already in the swarm. We assume the existence of a third-party membership service that can provide random subsets from the list of current participants. There exist techniques to implement such a membership service, like the central tracker used in BitTorrent, or by a distributed mechanism, either flood-based

(like Gnutella) or gossip-based [11]. Upon being queried for the first time by a joining peer, the membership service returns a random subset of m peers that are currently in the system. With the help of the membership service, the peer, including the source, tries to maintain concurrent connections to at least m neighbors. A peer’s neighbor list is continuously updated during the session to accommodate membership dynamics. When a peer departs, it notifies the membership service and its neighbors. When the number of neighbors connected to a peer becomes less than m due to departures, the peer will re-query the membership service again to replace departing neighbor(s) with randomly chosen new neighbors.

B. Buffering

The live stream is broken into small data unit fragments of fixed size. The source assigns monotonically increasing sequence numbers to the produced data items. We will call the sequence number of a data unit, the *index* of the data unit. The indices allow identification of the relative order of data units and are used in the management of the peer buffers. Peers are interested in what is being broadcast “now”, and therefore are loosely synchronized with the source. At any time, a peer is only interested in a small continuous window of the stream

depending on the latency between itself and the source. Thus only a small size buffer is needed to store the data units that are inside a peer's current window of interest. As shown in Fig. 5, a peer's window of interest scrolls at the same rate of the stream as the oldest data unit is being removed and later consumed by the media player. If a data unit is still missing upon its removal, it will cause playback discontinuity.

The buffer size determines the lag between a peer's playback point and the broadcast time at the source. A smaller buffer size produces a more "lively" stream but at the risk of causing higher data loss rate. On the other hand, a bigger buffer size improves the chances of obtaining data in time thus produces smoother playback. In this paper we assume that each peer can tolerate an one-minute lag behind the broadcast time and as such allocates a corresponding size of buffer. Note that old data units, outside the window of interest, are discarded from the buffer, hence the space requirements for a participating peer are fairly low, which could be construed as a further incentive for nodes to participate in the swarm.

When two peers establish a connection, they exchange bitfields representing the index of data units in their buffers. Each time a peer obtains a new data unit, it announces the availability to its neighbors by sending a *HAVE* message with the index number(s) of the newly downloaded unit(s). The overhead of exchanging *HAVE* messages is very small. Assuming each data unit contains one second of media data, a peer generates *HAVE* messages at an average rate of one message per second. Suppose the size of a *HAVE* message is 40 bytes, the upload/download bandwidth consumed by *HAVE* messages is only about 16kbps for a peer connected to 50 neighbors which is negligible compared to the stream bitrate. Finally, a newly connected peer defers playback until its buffer is filled up with contiguous data units to the average level of its neighbors. We define as *startup delay* as the delay between the time a peer connects and the time it actually begins playback.

C. A Credit-Based Incentive Scheme

Like many P2P applications, we allow users to configure how much upload bandwidth they are willing to contribute. Doing so allows other applications to share the same connection. It is not in the interests of the peer to set its upload bandwidth to the physical capacity (or to a higher value) because it will lead to congestion on the upstream link, which in turn (as per our proposed scheme) hurts the peer's download speed. Realizing that peers are heterogeneous in terms of upload capacity, we do not require everyone to contribute a minimum amount of bandwidth to be admissible into the system. Instead, the incentive scheme encourages resource-rich peers to contribute more upload bandwidth and subsidize resource-poor peers. Ideally, resource-poor peers are to be served normally as long as there are sufficient resources available. But once available resources become critical, the incentive scheme allows peers that contribute more upload bandwidth to receive better playback quality, thus encouraging them to continue their contribution.

A candidate solution is the tit-for-tat upload algorithm described in [7], which works well in real-world networks as shown by the success of BitTorrent. The algorithm works in a fully distributed way by letting each peer work independently to maximize its own download rate. A peer selects a fixed number of neighbors to upload while "choking" others. It decides which neighbors to upload based on the current

download rates it receives. As a result, peers that upload more tend to have higher download rates as well. Once a peer has finished downloading the whole file, depending on its level of altruism either departs or becomes a "seed" to continue uploading. The scheme has the important property of being *history-independent*. When a peer determines which neighbors to reciprocate, it does so without consulting the history of transactions or the long-term transfer rate. Instead only the instantaneous transfer rate is taken into account.

Adopting tit-for-tat in live streaming is slightly more complicated. In live streaming the average download rate a peer can achieve is restricted by the stream bitrate. Once a peer has obtained all the data units currently available in the system, it stops downloading until it slides its window of interest to accommodate a new data unit generated at the source. With a history-independent incentive scheme, an adversarial move during such transient periods would be to stop uploading as well because there is no *direct* prospect of reciprocity to continue uploading. By running some preliminary simulations we verified that such "selfish" behavior causes underutilization of resources and oscillations in download rates. To correct the situation, we need to encourage "seeding" behavior from (potentially adversarial¹) peers, we introduce a credit-based incentive that is designed to be more *history-dependent*. A peer assigns a credit (CR) to each neighbor to guide future reciprocation decisions. A neighbor's credit is continuously updated solely based on the net amount of data the peer has received from that neighbor. When a peer receives a data unit from neighbor i successfully (before playback), it increments neighbor i 's credit accordingly: $CR_i = CR_i + 1$.

With a history-dependent credit in effect, an adversarial peer cannot expect immediate rewards from short-term cooperative behaviors. Therefore it is encouraged to continue uploading to accumulate its credits regardless of the progress of its downloading. However, an ever increasing credit brings another problem: Having uploaded a certain amount of data to a neighbor, an adversarial peer may decide not to upload data anymore to that neighbor because it has accumulated enough credits to secure reciprocations in the future. Also in a dynamic environment, newly connected peers will find themselves in an adverse position to compete with peers that have been around for a while. To reduce old credits, we introduce an aging factor β into credit computation. Each peer periodically (default is once every second) updates its neighbor's credit by: $CR_i = \beta \times CR_i, 0 < \beta < 1$.

Each newly connected neighbor receives an initial credit $\epsilon \geq 0$. The value of ϵ determines the competitiveness of a new neighbor. Note that a large ϵ may undermine the effectiveness of the incentive scheme. In particular, an adversarial peer is tempted to *whitewash* itself [10] by constantly making new connections if ϵ is higher than its credit at current neighbors. In this paper, new neighbors receive an initial credit $\epsilon = 0$ to impose a penalty on new connections and discourage whitewashing. Because a peer has to build its credits from scratch, it is better off by staying with its current neighbors. Note that our incentive scheme is also resistant to cheating and collusion as peers make decisions strictly by observing the behaviors of their neighbors.

¹"Adversarial" in this context means a peer with selfish behavior tries to maximize its gains while still playing according to the rules of the P2P protocol, i.e., by exploiting weaknesses of the protocol.

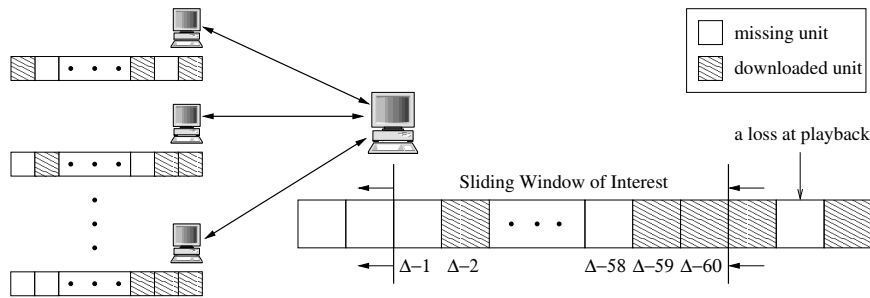


Fig. 5. A peer's sliding window (Δ is the index number of the latest data unit at the source).

D. Informed Push-Based Scheduling

A common feature of many multicast systems is that data delivery is scheduled in a push-based fashion. That is, to reduce overhead and accelerate propagation of delay-sensitive data, data are forwarded without explicit requests from the receivers. Push-based scheduling is more desirable for broadcasting live stream to a large set of dispersed nodes in a short time limit. However, there are a few disadvantages to pure push-based scheduling: (a) it is difficult to recover from data loss if the receiver is unable to ask for retransmissions, and (b) when there are multiple suppliers for a given receiver, uninformed push may create duplicate transfers, which is particularly undesirable for resource-constrained environments.

We devise an informed push-based scheduling strategy assisted by feedback from receivers. A peer always uploads to a fixed number of neighbors depending on its upload capacity². When it finishes uploading a complete data unit, it selects the next recipient based on its neighbors' credit rating: The neighbor with the highest credit will be chosen. Once a peer chooses a neighbor to reciprocate, it compares its own bit-field with the neighbor's bit-field to determine a subset of data units it is able to offer. The particular unit to be uploaded is then selected from the subset in a random fashion.

We do not employ a rarest-first selection policy because it tends to postpone transferring units imminent to playback since those units are usually more common in the system. We have determined through preliminary simulations that rarest-first policy causes unnecessary playback discontinuity. On the other hand, always selecting units imminent to playback yields better playback continuity in the short run, but can undermine data diversity in the system and eventually result in lower throughput because the performance of a swarm-based system is directly influenced by the data diversity available to peers. Therefore we choose to select the unit at random as a compromise between data diversity and playback continuity. With informed push, recovering from data loss caused by node failures is an automatic process. However, there is still a possibility that a peer receives duplicate data from multiple suppliers, especially when only a few data units are still missing from its buffer. When a duplicate transfer occurs, the receiver intervenes by sending back a `FEEDBACK` message, instructing the supplier to stop and instead send another data unit suggested in the feedback.

The source is handled as an exception. It receives no upload, so all its neighbors have a zero credit. To ensure fairness, the

source uploads to its neighbors in a round-robin fashion. Also the source always pushes out the latest data units with higher priority. This is done to facilitate in-time data delivery and avoid late data loss.

IV. PERFORMANCE EVALUATION

We developed a flow-level, discrete-event simulator to simulate the P2P live streaming network. The simulator models the propagation delay between peers but it does not model queuing delay, packet losses, or cross traffic. Without considering a physical network topology, we assume all nodes reside at the edge of the network and no bottleneck links exist in the core of the network. This simplification implies that congestion only happens at the access links to the network [4]. In a P2P streaming system, the bottleneck resources are indeed the limited upload capacity of end hosts. Therefore our simulator only models congestion at those outbound links, which is done by fairly dividing available bandwidth between concurrent flows leaving a node.

The simulated network consists of 1740 nodes, with a pairwise latency matrix derived from measuring the inter-node latencies of 1740 DNS servers using the King method [12]. We have also obtained similar results from simulations involving larger networks, where each node is a random pair of coordinates on a 2-D Euclidean space and the network delay between a pair of nodes is derived from their corresponding Euclidean distance. The average and maximum round-trip delay between node pairs in both the King data set and the Euclidean plane are approximately 180ms and 800ms respectively.

Based on the measurement studies of live streaming workloads, [29], [32], we model the inter-arrivals of newly joined peers by an exponential distribution, with average arrival rate λ . After a peer successfully joins the multicast, its sojourn time (session length) in the system is drawn from a log-normal distribution. The log-normal distribution captures the fact that a significant number of sessions are very short while at the same time its heavy tail represents peers that tune in for a much longer periods of time. Specifically, the median session length is only about 5.5 minutes while the mean length is around 15 minutes. We assume the source is broadcasting a constant bitrate stream at 512kbps. The stream consists of a sequence of media packets. Each packet contains the compressed media data spanning across a period of one second. Thus, loss of a single packet causes an one second playback discontinuity.

We divide peers into two categories according to their upstream bandwidth: 1) *resource-rich* peers have an upstream bandwidth of 2048kbps, which is four times the bitrate; 2) *resource-poor* peers have a constrained upstream bandwidth of 256kbps, half the bitrate. The percentage of resource-rich

²We assume that the underlying transport protocol is able to saturate the upload capacity and control congestion as the same time. In practice, a variant of TCP Friendly Rate Control (TFRC) protocol [15] may be used instead of TCP to avoid the abrupt changes of the sending rate, which are unfavorable in a streaming application.

TABLE I
DEFAULT SIMULATION PARAMETERS.

Concurrent peer connections	$m = 20$
Average arrival rate	$\lambda = 1$ arrival per second
Credit aging factor	$\beta = 0.9$ per second
Initial credit	$\epsilon = 0$
Startup delay	30 seconds
Size of the sliding window	60 seconds
Source capacity	2560 kbps

peers α controls the resource index RI and RI_{tree} in the system as: $RI = 3.5\alpha + 0.5$ and $RI_{tree} = 4\alpha$. Since we only consider congestion at upstream links, peers are assumed to have unlimited downstream capacity. We essentially assume that the download capacity is greater or equal to the upload capacity at each peer node, allowing us to capture asymmetric access schemes as well, e.g., xDSL. Another assumption is that the download capacity must be higher than the stream bitrate. This is needed to allow a peer to receive the full quality stream while not saturating its downstream link. (Clearly, peers with download capacity less than the stream bitrate, cannot sustain receiving the streamed data anyway, regardless of the streaming technique used.) In the steady state, the average download throughput of a peer is equal to the stream bitrate, and because this value is less than the download capacity, the downstream link is not the bottleneck. Hence, there is no need to model the download capacity of the peers.

To compare the performance of the swarm-based live streaming protocol with a tree-based one, we implemented a simplified version of the tree overlay protocol borrowed from [28]. Like many existing P2P live streaming approaches, it builds and maintains a single connected tree rooted at the source. When a peer needs to connect to the tree, either at its join time or at the time of a reconnection, it contacts the peer membership service to get a random subset of m peers that are currently in the system. It will then probe the m peers to see if they are currently connected to the tree and have enough capacity to support a new child. If there are multiple positive replies, the peer selects the parent with the minimum depth. This process is repeated until the peer is able to find a parent and connect to the tree. Unlike [28], we do not give higher priority to join requests from potential contributors based on the reasoning that it is very difficult to realize such a preemption scheme in real-world networks (see Section 2). Therefore there may be cases where the tree becomes prematurely saturated, leaving some resource-rich peers unconnected and unable to contribute their upload bandwidth. We also employ a different tree repairing method when an internal node fails: Instead of having each disconnected descendant look for a new parent independently, only the children of the failed node try to reconnect while others stay put in the subtrees. Such graft-like method generates fewer reconnection requests. It is also supposed to favor the creation of more stable tree structures.

Unless otherwise noted, each run of simulation simulates a period of two hours. To allow the system to reach steady state behavior, we start to collect statistics after a warm-up period of one hour in simulation time. Default simulation parameters are listed in Table I. We evaluate the performance of the protocols using two main metrics: *average stream quality* and *aggregate system throughput*. Average stream quality measures the playback continuity received at individual peers, defined as the fraction of data units that arrived before their playback over the total number of units. Aggregate system throughput is the

sum over all nodes of the instantaneous download throughput, normalized by the total demand ($\#nodes \times$ stream bitrate).

A. Performance of Tree-Based Protocol

Fig. 6 plots the average stream quality as a function of α , the percentage of resource-rich peers. It can be seen that the stream quality remains very low when there are few resource-rich peers in the system. Only after α exceeds 0.25, the stream quality begins to improve to an acceptable level. When half of the participants are resource-rich peers, the tree is able to deliver a near perfect quality stream to all participating peers. Fig. 6 also shows that the source's capacity is very important in a tree-based protocol. A small fan-out degree at the source has a big negative impact on the performance because the tree is more likely to become unbalanced if a few resource-poor peers happen to be connected near the source.

Fig. 7 shows the changes in the tree's aggregate throughput over time under different resource conditions. We find that lack of resources causes instability in the tree. When $\alpha = 0.4$, the system is able to quickly recover from node failures. With a smaller $\alpha = 0.3$, the system becomes more vulnerable to peer transience. Oscillations in throughput occur more frequently and last longer. When $\alpha = 0.25$, the tree begins to experience large-scale and continuous breakdowns. The throughput plunges when a lot of peers are disconnected due to departures occurring near the source. Worse, if the tree becomes prematurely saturated, some breakdowns last for very long. The resulting throughput fluctuation is destructive to playback continuity (frequent buffer underflows).

Fig. 8 shows the underutilization of resources in the tree at $\alpha = 0.25$ over time. Note that the throughput curve in Fig. 7 closely follows the tree's capacity in this figure, indicating that our tree-based protocol is efficient in locating unsaturated nodes (if they exist). The difference between RI and RI_{tree} demonstrates the inherent underutilization of resources in the tree-based protocol because resource-poor peers are unable to contribute. The capacity of the tree is determined by the amount of upload bandwidth from resource-rich peers that are *currently* in the tree. Apparently the tree's capacity is maximized and equal to RI_{tree} when all resource-rich peers are connected. However, as shown in Fig. 8, the tree's capacity is only maximized in a few cases. Instead, it often happens that many resource-rich peers are *blocked* from joining the prematurely saturated tree, which exacerbates the underutilization of resources.

To further illustrate the instability of the tree structure, we plot in Fig. 9 the cumulative distribution of disruption periods. A disruption period is the interval between the time a peer is disconnected from the tree and the time it finds a new parent, during which the peer suffers playback discontinuity. Fig. 9 confirms that a tree is much more unstable in a resource-constrained environment. When $\alpha = 0.4$, most disruptions are recovered in less than one second; but when α drops to 0.25, more than 30 % of disruptions last longer than ten seconds.

B. Performance of Swarm-Based Protocol

Figs. 10 and 11 provide a comparison against Figs. 6 and 7. Comparing Figs. 10 and 6, we find that the swarm-based protocol is more resilient to scarcity of resources because a swarm can fully utilize all available resources. The average stream quality in Fig. 10 quickly improves as RI increases.

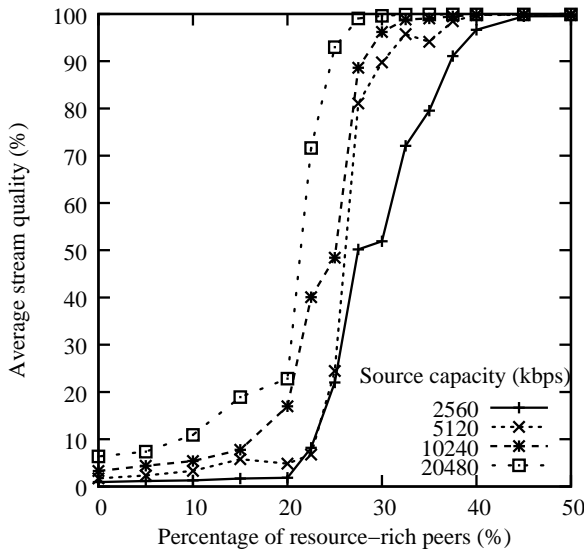


Fig. 6. Tree-based stream quality vs. α .

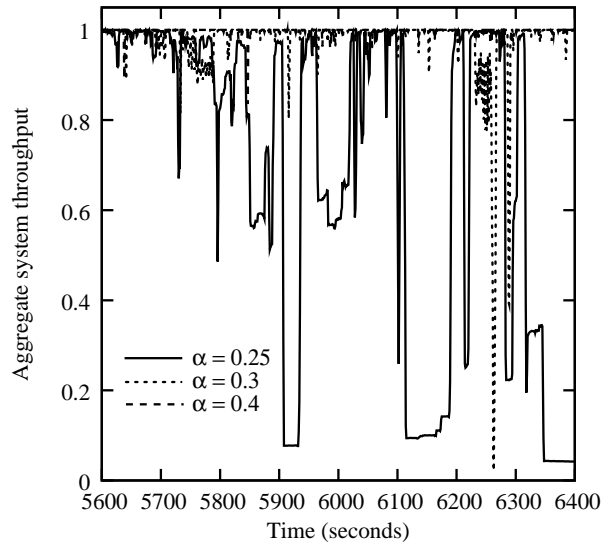


Fig. 7. Time-varying system throughput.

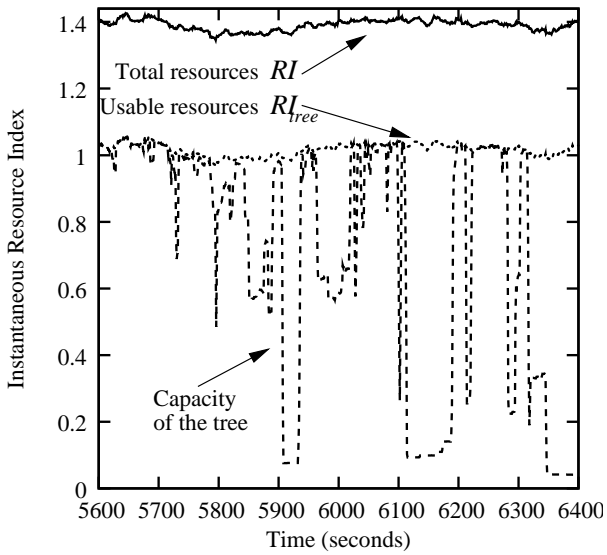


Fig. 8. Available vs. used resources.

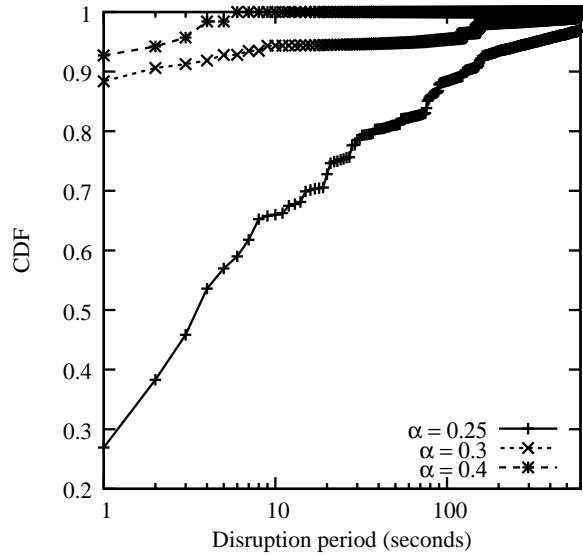


Fig. 9. CDF of disruption periods.

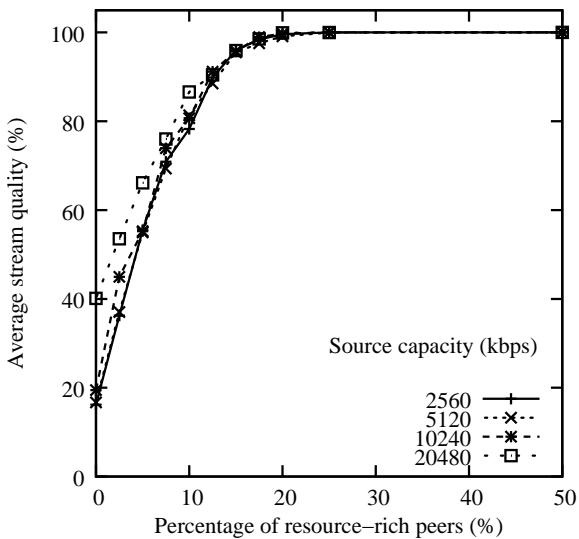


Fig. 10. Swarm-based stream quality vs. α .

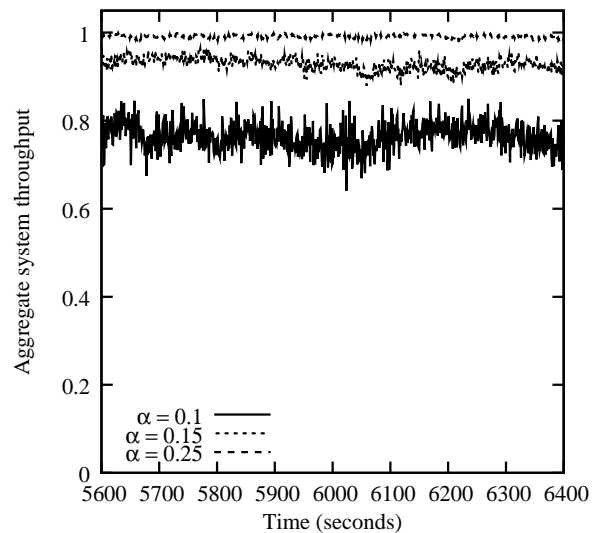


Fig. 11. Time-varying system throughput.

At $\alpha = 0.2$, the swarm-based system is able to deliver a near-perfect stream while the stream in the tree-based system is unwatchable (quality $\ll 50\%$). We also find that a swarm requires less source capacity to yield good performance. Indeed, in a tree, data forwarding is performed only by a fraction of peers, while in a swarm all the peers are engaged in distributing data. The problem of premature saturation, which plagues a tree with limited fan-out degree at the root, seldom happens in a swarm. A swarm, not only produces better stream quality, but is also more stable than a tree in the presence of churn. The reason is that a swarm does not need to maintain a rigid structure. Instead the propagation path of data units is constantly changing to adapt to network dynamics. Fig. 11 plots the swarm's aggregate throughput over time. When $\alpha = 0.25$, the system throughput remains close to 1.0 with a sample standard deviation, σ , of only 0.003. This is in sharp contrast with Fig. 7, where for $\alpha = 0.25$, σ is 0.36.

1) *Scalability*: The scalability of the swarm-based protocol is studied by increasing the arrival rate λ , thus increasing the average network size (average number of peers). Fig. 12 shows how the average and the maximum path length to deliver data units from the source to all participating peers is affected. The path length grows in an almost logarithmic fashion for increasing number of nodes. This is expected because each generated data unit propagates along a spanning tree rooted at the source, whose depth grows logarithmically with the size of the network. Even with tens of thousands of nodes in the swarm, the propagation tree depth remains reasonably low, incurring a maximum end-to-end latency in the order of a few seconds (ignoring waiting at each hop). We also found that the average stream quality is not affected by the network size in a noticeable way, and we omit the results for brevity. Moreover, because a peer is connected to a fixed number of neighbors regardless of the total number of nodes, the overhead at each peer is independent of network size.

2) *The Effect of Startup Delay*: When a peer joins the system, its buffer is empty and its credit is zero. Therefore the data loss rate is likely to be very high before the peer can obtain a few data units and accumulate some credits. Fig. 13 shows the average data loss rate during a peer's startup time under different resource conditions. With a reasonable resource index ($\alpha = 0.25$), a newly connected peer can expect its buffer to be filled up to the average level in less than 30 seconds by its neighbors. From that point on, the peer is able to receive the stream with nearly no data loss. Therefore a 30-second startup delay is enough for a new peer to start smooth playback. However, we also find that a much longer startup delay is necessary if the resource index gets lower. Such a penalty on newcomers can be partly lifted with a higher initial credit ϵ .

3) *Providing Incentives*: To investigate the effect of the credit-based incentive scheme, we compare, in Fig. 14, the average stream quality received by resource-rich and resource-poor peers. As expected, the incentive scheme works in an adaptive way, responding differently to different resource conditions in the system. When the resource index is larger than 1.0, both resource-rich and resource-poor peers are able to receive near-perfect stream quality because there are more than enough resources to sustain the whole population. However, once the resource index drops below 1.0, the effect of the incentive scheme begins to show up: the stream quality received by resource-poor peers plunges, while resource-rich peers are not affected as much. This difference in quality of service is

more distinguishable if the resource index continues to drop. Therefore, resource-rich peers are encouraged to contribute larger amount of uplink bandwidth, because doing so will lead to better playback quality in the presence of resource variations. Another beneficial effect of the incentive scheme is when the system is short of resources, resource-poor peers are more likely to depart due to poor stream quality, while resource-rich peers are more likely to stay since they see much less quality degradation. As a result, the system will be able to self-recover from resource shortage and restore the average stream quality.

Fig. 15 plots the average path length from source to resource-rich and resource-poor peers as a function of α . It is desirable to receive data across shorter paths, since it reduces the propagation delay and the probability of service disruption. We find that there is clear correlation between the path length to a peer and the amount of its contributions. Resource-rich peers receive data in fewer hops, which serves as another incentive for them to contribute more uplink bandwidth. This is a natural consequence of the credit-based incentive scheme that favors peers with higher credits. A resource-rich peer accumulates more credits at its neighbors by uploading more. Therefore it is able to "preempt" other, resource-poor, peers in the contention for upload slots. Since resource-rich peers have higher out-degree, their placement closer to the source reduces the depth of the tree.

V. RELATED WORK

Based on the observation that a small number of peers stay in the system for very long periods of time, the authors of [28] claimed that there is *inherent stability* in a single-tree architecture. They found that simple algorithms such as the minimum-depth algorithm is able to provide good stability by minimizing the depth of the tree. Our results complement their findings by showing that the single-tree overlay can indeed achieve good performance *provided* there are sufficient bandwidth resources. However, we also demonstrate that because of its susceptibility to premature saturation, a tree is unstable in resource constrained environments.

Proactive approaches that look for extra "backup" parents beforehand have shown in [27], [34] to have much better resilience to node failures than traditional reactive recovery methods in a tree. However, backup paths are not always available in P2P streaming systems due to the high bandwidth requirement. Under constrained resource conditions, there is barely enough bandwidth to support the current number of nodes, which makes it difficult or infeasible to find extra suppliers.

To address the unfairness and vulnerability issues of a single-tree overlay, a few solutions advocating multiple-tree models have been proposed in SplitStream [3] and CoopNet [5]. The media stream is encoded into k descriptions (also known as stripes or layers), each distributed across an independent tree. The vast majority of peers are interior nodes in only one tree, which improves fairness. Also a node failure only causes the loss of one single stripe on average. A multiple-tree based approach is tightly coupled with advanced coding techniques like multiple description coding (MDC), which are not widely available yet. In order to receive the complete stream, a peer must join k trees, requiring higher control overhead for maintaining and repairing each tree. The *Bit-for-Bit* incentive scheme in the multiple-tree model does not work well in a heterogeneous environment either, where many

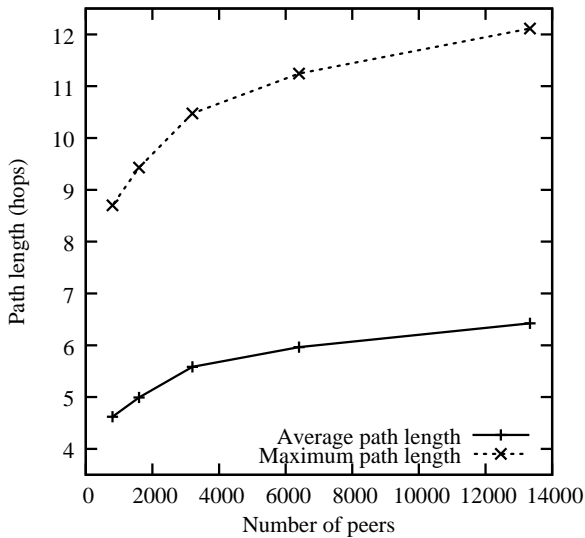


Fig. 12. Swarm scalability ($\alpha = 0.25$).

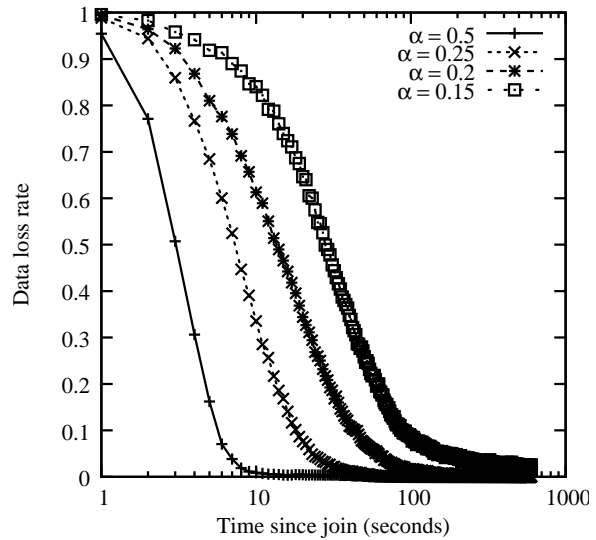


Fig. 13. The effect of startup delay.

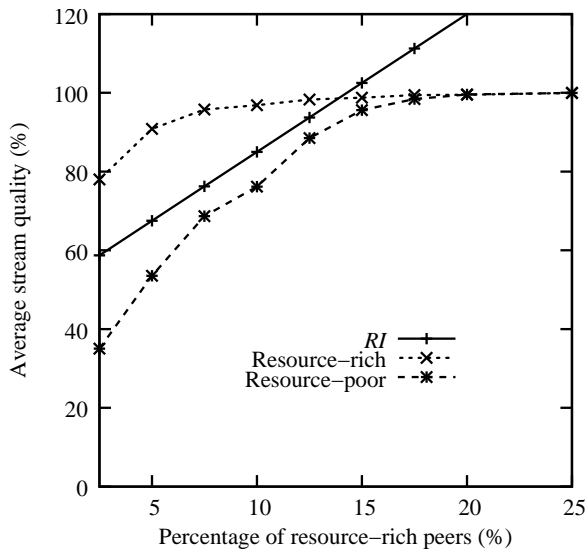


Fig. 14. Service differentiation.

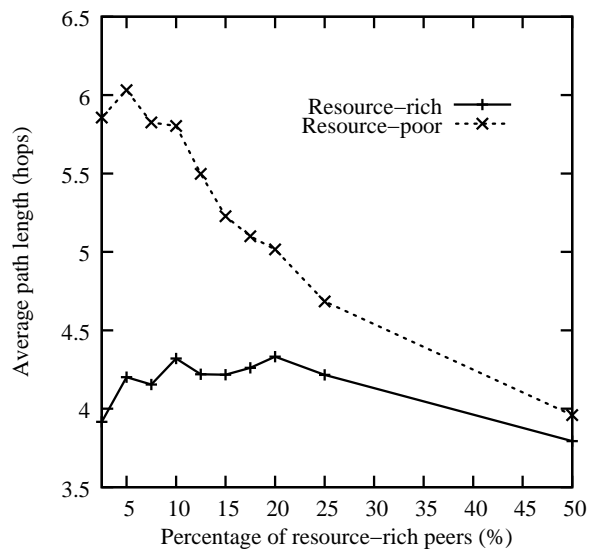


Fig. 15. Peer location.

users with asymmetrical connections cannot upload as fast as they download.

Bullet [17] combines a standard single-tree structure with an overlay mesh layered on top of the tree. Bullet nodes self-organize into an overlay tree, which is used to convey control messages and most data packets. A highly variable mesh structure is then constructed among peers via random connections that are orthogonal to the tree. The mesh overlay enables nodes to quickly locate multiple peers and retrieve missing data items in parallel. Thus Bullet is able to avoid bandwidth bottlenecks in the tree by allowing leaf nodes to forward data as well. However, since it still relies on the tree to disseminate most data, Bullet does not completely address the inherent issues in the single-tree structure. Also there are no incentives for Bullet nodes to contribute resources.

In another mesh-based P2P streaming approach, e.g., the Peer-to-Peer Receiver-driven Mesh-based Streaming (PRIME) protocol proposed in [20], media data delivery is accomplished within two phases: the push-based diffusion phase based on the diffusion tree and the pull-based swarming phase from leaf nodes to the intermediate nodes or other leaf nodes in the diffusion tree. Notably, in mesh-based P2P streaming each participating peer node can have multiple parent nodes, which

collectively fill its incoming link bandwidth, delivering as much sub-streams as possible to the given node. A mesh-based overlay is easy to maintain and it is pretty resilient to rapid node joins/leaves. In [18], [19], [20], it is also assumed that in the final overlay structure any directly connected peer node pair should have roughly the same bandwidth capacity for each flow. The value of bandwidth per flow indicates the relationship between incoming/outgoing link bandwidth and the incoming/outgoing degrees at each peer node.

Inspired by the success of BitTorrent similar approaches have been proposed to support live streaming in an unstructured mesh overlay, namely Coolstreaming [35] and Chainsaw [23]. Our approach shares the same motivation with them. However, both Coolstreaming and Chainsaw assume a cooperative environment where nodes upload willingly. Thus no incentives are provided to justify contributions. They also employ a different pull-based scheduling strategy to disseminate delay-sensitive data.

One of the challenges in P2P networks is the selfish behavior of uncooperative peers [21]. In the P2P streaming settings, a good incentive mechanism is indispensable for the system to encourage peer nodes to increase their contributions. Recent work in [2], [14], [21], [31] also presents different approaches

for incentive mechanisms in P2P streaming. The bottom line is that the system has to be contribution-aware [31] and/or reputation-aware [13], [16] in the first place.

In [2], a game theoretic framework is presented for incentives in P2P systems and the incentives are accomplished mainly by selective denial of requests. In [31], the system distributes more bandwidth to nodes that make more contributions. In [21], an incentive mechanism is proposed aiming at increasing scalability and stream quality as well as decreasing the number of tree disruptions. The work in [14] considers incentives for cooperation through service differentiation.

In summary, our approach differs from existing methods in the following aspects: (a) it allows total freedom (and virtual zero maintenance overhead) by allowing peers to self-organize into an unstructured overlay in an ad-hoc fashion, unlike [17], [20] which assume (and hence need to maintain) either a tree-based diffusion or a mesh-based overlay swarm; (b) it utilizes a credit-based incentive scheme which operates well in both cooperative and un-cooperative environments, meaning no particular assumptions about the cooperative (or not) nature of peers needs to be made, and (c) it employs an incentive scheme which operates exclusively on local information (hence, does not require auxiliary reputation etc., protocols) and it is more history-dependent than the tit-for-tat method employed in BitTorrent-like schemes [1], [7].

VI. CONCLUSIONS AND FUTURE WORK

We have presented the design of an incentive-aware live streaming system which does not maintain a rigid structure. Instead, nodes self-organize in an ad-hoc fashion into a random-graph overlay. An informed push-based scheduling strategy is employed to propagate data efficiently without many duplicate transfers. Our simulation results show that the architecture can scale to a large number of nodes and is resilient to high rate of churn. With the help of a credit-based incentive scheme, our system is able to work in a non-cooperative environments by rewarding contributors while penalizing non-contributing nodes. The main results of this study are that the swarm scheme outperforms comparable tree-based schemes but also provides performance dividends exactly when necessary, i.e., when resources are scarce. In fact, it is under resource-scarce conditions that the incentive scheme provides resource-affluent nodes with a performance advantage that encourages their continued presence and participation.

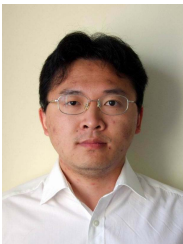
We are currently studying the impact that cross-traffic and flash crowds can have on the scheme. More interesting questions are those that need to be addressed towards an implementation of the scheme. For example, an important issue is how to make the swarm topologically aware so that the data propagation path is efficient in terms of metrics such as link stress and end-to-end latency. It is also important for recipients to be able to verify data integrity on the fly during a P2P live streaming session. However, existing protocols to enforce data integrity are either expensive or inapplicable to P2P streaming. Among the explored possibilities are reputation-based schemes [13], [16], and in general schemes that do not have to rely on pre-existing trust relationships. We intend to investigate these issues in our future work.

REFERENCES

- [1] BitTorrent: <http://www.bittorrent.com/>.
- [2] C. Buragohain, D. Agrawal, and S. Suri, "A Game-Theoretic Framework for Incentives in P2P Systems", in *Proc. IEEE P2P*, Linköping, Sweden, September 2003.

- [3] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth Multicast in Cooperative Environments", in *Proc. ACM SOSP*, Sagamore, NY, USA, October 2003.
- [4] Z. Cataltepe and P. Moghe, "Characterizing Nature and Location of Congestion on the Public Internet", in *Proc. IEEE ISCC*, Antalya, Turkey, July 2003.
- [5] P. A. Chou, V. N. Padmanabhan, and H. J. Wang, "Resilient peer-to-peer streaming", Technical Report MSR-TR-2003-11, Microsoft Research, 2003.
- [6] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast", *IEEE Journal of Selected Areas in Networking*, vol. 20, no. 8, October 2002, pp. 1456-1471.
- [7] B. Cohen, "Incentives build robustness in BitTorrent", in *Proc. Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, USA, June 2003.
- [8] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming Live Media over a Peer-to-Peer Network", Technical Report, Stanford University, 2001.
- [9] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment Issues for the IP Multicast Service and Architecture", *IEEE Network*, vol. 14, no. 1, January/February 2000, pp. 78-88.
- [10] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-Riding and Whitewashing in Peer-to-Peer Systems", in *Proc. SIGCOMM PINS*, Portland, USA, August 2004.
- [11] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie, "Peer-to-Peer Membership Management for Gossip-Based Protocols", *IEEE Transactions on Computers*, vol. 52, no. 2, February 2003, pp. 139-149.
- [12] K. Gummadi, S. Saroiu, and S. Gribble, "King: Estimating Latency between Arbitrary Internet End Hosts", in *Proc. ACM SIGCOMM IMW*, Marseille, France, November 2002.
- [13] M. Gupta, P. Judge, and M. Ammar, "A Reputation System for Peer-to-Peer Networks", in *Proc. ACM NOSSDAV*, Monterey, USA, June 2003.
- [14] A. Habib and J. Chuang, "Incentive Mechanism for Peer-to-Peer Media Streaming", in *Proc. IEEE IWQoS*, Montreal, Canada, June 2004.
- [15] M. Handley, S. Floyd, J. Pahlke, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", *RFC 3448*, 2003.
- [16] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The Eigentrust Algorithm for Reputation Management in P2P Networks", in *Proc. ACM WWW*, Budapest, Hungary, May 2003.
- [17] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh", in *Proc. ACM SOSP*, Sagamore, NY, USA, October 2003.
- [18] N. Magharei and R. Rejaie, "Understanding Mesh-based Peer-to-Peer Streaming", in *Proc. ACM NOSSDAV*, Newport, USA, May 2006.
- [19] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches", in *Proc. IEEE INFOCOM*, Anchorage, USA, May 2007.
- [20] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-driven Mesh-based Streaming", in *Proc. IEEE INFOCOM*, Anchorage, USA, May 2007.
- [21] D. Manzano and N. da Fonseca, "An Incentive Mechanism for Peer-to-Peer Networks with Live Streaming", in *Proc. IEEE GLOBECOM*, San Francisco, USA, November 2006.
- [22] W. T. Ooi, "Dagster: Contributor-Aware End-Host Multicast for Media Streaming in Heterogeneous Environment", in *Proc. MMCN*, San Jose, USA, January 2005.
- [23] V. Pai, K. Tamilmani, V. Sambamurthy, K. Kumar, and A. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast", in *Proc. IPTPS*, Ithaca, USA, February 2005.
- [24] G. Pandurangan, P. Raghavan, and E. Upfal, "Building Low-Diameter P2P Networks", in *Proc. ACM STOC*, Heraklion, Greece, July 2001.
- [25] S. Saroiu, K. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems", in *Proc. MMCN*, San Jose, USA, January 2002.
- [26] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol", in *Proc. IEEE INFOCOM*, Hong Kong, China, March 2004.
- [27] J. Silber, S. Sahu, J. Singh, and Z. Liu, "Augmenting Overlay Trees for Failure Resiliency", in *Proc. IEEE GLOBECOM*, Dallas, USA, November/December 2004.
- [28] 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. ACM SIGCOMM*, Portland, USA, August/September 2004.
- [29] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An Analysis of Live Streaming Workloads on the Internet", in *Proc. ACM SIGCOMM IMC*, Taormina, Italy, October 2004.
- [30] D. Stutzbach, D. Zappala, and R. Rejaie, "The Scalability of Swarming Peer-to-Peer Content Delivery", in *Proc. IFIP Networking*, Waterloo, Canada, May 2005.

- [31] Y. Sung, M. Bishop, and S. Rao, "Enabling Contribution Awareness in an Overlay Broadcasting System", in *Proc. ACM SIGCOMM*, Pisa, Italy, September 2006.
- [32] E. Veloso, V. Almeida, W. Meira, A. Bestavros, and S. Jin, "A Hierarchical Characterization of a Live Streaming Media Workload", in *Proc. ACM SIGCOMM IMW*, Marseille, France, November 2002.
- [33] V. Vishnumurthy and P. Francis, "On Random Node Selection in P2P and Overlay Networks", Technical Report, Department of Computer Science, Cornell University, 2004.
- [34] M. Yang and Z. Fei, "A Proactive Approach to Reconstructing Overlay Multicast Trees", in *Proc. IEEE INFOCOM*, Hong Kong, China, March 2004.
- [35] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "DONet/CoolStreaming: A Data-driven Overlay Network for Live Media Streaming", in *Proc. IEEE INFOCOM*, Miami, USA, March 2005.



Tianhao Qiu earned a M.Sc. degree in Computing Science from the University of Alberta as well as a M.E. degree in Information Engineering from Beijing University of Posts and Telecommunications (BUPT) in China. Tianhao Qiu works as a software engineer at Fortinet Technologies (Canada) Inc. Prior to joining Fortinet, he has worked at Eyeball Networks in Vancouver and Sun Microsystems in Beijing, China.



Ioanis Nikoladis is an Associate Professor with the Computing Science Department at the University of Alberta. He received his B.Sc. from the University of Patras, Greece, in 1989 and his M.Sc. and Ph.D. in Computer Science from Georgia Tech in 1991 and 1994, respectively. Between 1994 and 1996 he worked for the European Computer-Industry Research Center in Munich, Germany, in the area of distributed computing. He joined the University of Alberta in January 1997. He has published more than sixty articles in books, journals, and conference proceedings in the area of computer networking. His research interest range from network modeling and simulation, to large scale data delivery systems, to mobile and secure networking. Since 1999 he has been a member of the editorial board (and is currently the Editor in Chief) of the IEEE Network magazine, where he was also the column editor for New Books & Multimedia until 2006 and the guest co-editor of the special issue on "Web Performance". Since 2000 he has been a member of the editorial board for the Computer Networks journal (Elsevier). He guest co-edited the Computer Networks journal special issue on "Wireless Local Networks". He has served in the technical program committees of numerous conferences, including ICC, Globecom, INFOCOM, LCN, IPCCC, PerCom, IFIP Networking, and CNSR. He is in the steering committee of WLN (co-located annually with IEEE LCN) and in the steering committee of the ADHOCNOW conference. He was the conference co-chair of ADHOCNOW 2004. He is a member of IEEE and ACM.



Fulu Li is currently a Ph.D. student in the Media Lab at MIT. He has a wide range of research interests in both fundamental and applied data-oriented areas including the construction of adaptive, resilient wireless communication systems, combinatorial optimization algorithms, cooperative routing, cooperative security, media streaming and bioinformatics, etc. Fulu is a recipient of National Outstanding Student award during his undergraduate study. Fulu also received a Gold Pride award from Nortel Networks Inc., a Bravo award from Motorola Inc., a student travel award from NSF. Fulu was a Motorola fellow from 2004 to 2006. Fulu received a best student paper award at IEEE WTS '2007. He received his M.Sc. degree in Computing Science from the University of Alberta.