Outreach: Peer-to-Peer Topology Construction towards Minimized Server Bandwidth Costs

Tara Small, *Member, IEEE,* Baochun Li, *Senior Member, IEEE,* and Ben Liang, *Member, IEEE*

***Abstract*— On-demand and live multimedia streaming applications (such as Internet TV) are well known to utilize a significant amount of bandwidth from media streaming servers, especially as the number of participating peers in the streaming session scales up. To scale to higher bit rates of media streams and larger numbers of participating peers, overlay tree or mesh topologies are typically constructed, such that peers utilize their available upload capacities to alleviate the excessive bandwidth demands on stream servers. Previous works rely on random selections of upstream peers, without optimizing the topologies towards maximized utilization of peer upload bandwidth, and as a result, minimized bandwidth costs on streaming servers.**

We propose *Outreach*, a distributed algorithm to construct overlay topologies among participating peers in streaming sessions. The design objective of *Outreach* is to optimize the quality of overlay topologies towards *scalability*, with respect to the number of participating peers in the session. To be scalable, Outreach seeks to maximize the utilization of available upload bandwidth on each participating peer, and consequently minimize the total bandwidth costs on streaming servers. With analysis, we show that Outreach constructs topologies such that peers can fully utilize their upload capacities, and present a practical distributed algorithm. With simulation-based comparison studies, we show that Outreach effectively achieves its goals in a high-churn peer-to-peer network with an assortment of peer uplink capacities and link delays.

***Index Terms*— Peer-to-peer multimedia streaming, overlay topology construction, server bandwidth optimization.**

I. Introduction

“The media streaming server is not responding.” Frequently, we are greeted with messages such as this when attempting to open a multimedia stream from a media streaming server. The streaming server is overloaded with an overwhelming number of simultaneous streams that it is serving. Due to the bandwidth-intensive nature of multimedia streams, it is challenging and *costly* to deploy streaming servers with sufficient uplink bandwidth to satisfy the needs of a large number of users. For example, it may take months to deploy an OC-12 uplink (622 Mbps), and it costs around $25,000 a month to operate the link. With a typical streaming bit rate of 800 Kbps for near-DVD video quality, even an OC-12 link is saturated with only 800 users.

To scale to a larger number of simultaneous users (possibly up to millions) in a multimedia streaming session without overwhelming the streaming servers, it has been proposed that a *peer-to-peer* overlay be constructed at the application layer. The main advantage of the peer-to-peer communication paradigm is obvious: each peer contributes its own uplink bandwidth to assist streaming to other peers in the same session, thus alleviating the bandwidth load on dedicated streaming servers. The corresponding disadvantage is that peers are *transient* in nature: they join and leave the session arbitrarily and unpredictably. The advantage of reduced server load clearly outweighs the disadvantage of transient peers, leading to the conclusion that peer-to-peer streaming offers compelling benefits to be implemented in real-world applications. Several emerging peer-to-peer streaming implementations, such as *CoolStreaming* [1], have clearly supported this observation.

In contrast to its potential benefits, most of the existing work in peer-to-peer streaming failed to offer sufficient insights towards the construction of peer-to-peer overlay topologies that *maximize* peer bandwidth contributions, and consequently minimize the load on dedicated servers. We argue that fully decentralized algorithms that construct such topologies are critical towards the scalability of peer-to-peer streaming sessions, to potentially serve millions of peers with minimal server bandwidth costs.

In this paper, we present *Outreach*, a peer-to-peer topology construction algorithm that seeks to achieve such an objective of *scalability*. Our original contributions are the following. *First*, we develop a rigorous analytical framework that minimizes server bandwidth cost in an ideal environment with complete knowledge of all participating peers in the streaming session. Such a framework allows peers to select “neighbors” in an optimal fashion, so that peer upload bandwidth is most effectively utilized. *Second*, based on our analytical insights, we further present the *Outreach* algorithm to construct topologies based on local knowledge, in a fully decentralized manner. Since strict optimality is not possible with transient peers, we use the average bandwidth cost at the streaming server as our optimization objective. *Finally,* we show that Outreach is resilient to churn with arbitrary distributions of peer lifetimes.

The remainder of this paper is organized as follows. In Sec. II, we first highlight our contributions in the context of related work. The peer-to-peer network model that we consider is presented in Sec. III. In Sec. IV, we present our analytical framework that provides important insights towards the design of the Outreach algorithm. We present the decentralized Outreach algorithm in Sec. V. We evaluate the performance of Outreach in Sec. VI. Finally, we conclude the paper in Sec. VII.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada and Bell Canada’s support through its Bell University Laboratories R&D program.

The authors are with the Department of Electrical and Computer Engineering, University of Toronto (emails: {tsmall@eecg, bli@eecg, liang@comm}.toronto.edu).

II. Related Work

We categorize existing work towards topology construction for peer-to-peer streaming into two groups: *tree-based* and *random-neighbor* topologies.

Tree-based topologies in peer-to-peer streaming, such as Coopnet [2] and Splitstream [3], are vulnerable to high "churn" (departure) rates of peers, since the departure of one peer affects all of its children. Algorithms such as NICE [4] and ZIGZAG [5] propose repair algorithms and use hierarchical clustering to keep the control overhead low while scaling to hundreds of peers. Such maintenance algorithms are complicated and may not be effective in a high-churn environment. Due to the transient nature of participating peers in a streaming session, we believe that tree-based topologies are not particularly suitable for peer-to-peer streaming.

In contrast, Bullet [6] constructs mesh-based topologies that are based on tree topologies where each peer distributes its received data messages among its children with the most spatial diversity possible. Higher throughput is achieved than using tree topologies alone, since peers locate messages they have not yet received in the peer-to-peer session, and the downloading processes proceed in parallel. Bullet emphasizes that a high aggregate upload bandwidth is needed in the downloading session, due to the transient nature of peers and heterogeneous peer upload capacities. However, the mesh topologies constructed by Bullet are still quite rigid, as they are based on trees. We concur that mesh-based topologies are superior with respect to the total available uplink capacities, and seek to construct the best possible mesh to minimize bandwidth costs on streaming servers.

Several recent papers, such as CoolStreaming [1], PRM [7], GridMedia [8], and Chunkyspread [9], have proposed to "spread" data to *randomly* chosen neighbors, using either "push" or "pull" techniques. While these techniques are robust to dynamic changes (churn), peers can only improve their utilization using local information; therefore, these strategies do not choose neighbors to globally optimize any particular metric, such as bandwidth costs at the servers.

With respect to upload and download bandwidth capacities of participating peers, BitTorrent [10] assumes that the uplink and downlink bandwidth of peers are symmetric, and requires altruistic peers to contribute their uplink bandwidth if this assumption is not valid. CoolStreaming [1] assumes that a peer may be served by the uplink of any *one* of its many neighbors. ChunkySpread [9] assumes that peers are fair in that they provide as much uplink bandwidth as their downlink bandwidth. We firmly believe that peers have *asymmetric* uplink and downlink bandwidth capacities (with considerably smaller uplink capacities), as they are mostly served by ADSL or cable broadband connections. Each peer may receive data from multiple upstream peers, and the gap between the total available download and upload capacities in the streaming session should be bridged by dedicated streaming servers. Such views are shared by PROMISE [11] and PALS [12]. Our objective is to minimize the bandwidth required from these dedicated servers.

III. Outreach: Preliminaries

A peer-to-peer streaming session (or a peer-to-peer "*network*") is a collection of peers that rely not only on a small number of multimedia streaming servers to receive their data messages, but also on the uplink bandwidth of the participating peers themselves. Each participating peer contributes its uplink bandwidth to serve other peers, relieving the burden that would otherwise be imposed on the dedicated streaming servers. In the streaming session, peers must receive data messages at a constant bit rate in order to guarantee high-quality playback. While our discussions in this paper are focused on peer-to-peer streaming applications, the constructed topologies apply equally well in content distribution sessions of bulk data (*e.g.*, file downloading).

Without loss of generality, we assume that the peer-to-peer network consists of *one* multimedia source (known as "the server") and $N - 1$ *peers*. In the case of multiple streaming servers, they are functionally equivalent to one server with the same total uplink bandwidth. Our goal is to stream media from the server to each of the participating peers, to maximize the peer upload bandwidth used so that we minimize the cost at the dedicated server for a given number of peers, and consequently maximize scalability. We assume that peers can upload at a random rate U, and that the media must be played back at a fixed rate p. Values of U are sampled from a distribution that accounts for varying peer upload capabilities at the peers. The number of peers N may vary over the duration of the experiment, since peers are more likely to participate in the streaming session at peak time periods in a day or a week.

Typically, in a mesh-based peer-to-peer topology, "availability maps" of buffered messages that have been recently received by a peer are advertised to its neighbors, who may subsequently request specific messages, depending on their rarity and playback deadlines. We assume that recent coding techniques (*e.g.*, *network coding* [13]) are used, rather than advertisements of availability. With network coding, for example, peers could compute and send linear combinations of previously received messages with randomly chosen coefficients as in [14] and [15]. Since the original data message can be decoded from a number of coded messages that consist of independent linear combinations, all coded messages are equally useful when received, and there is no need to "pull" or "push" the rarest message, or to exchange availability maps. We assume that such state-of-the-art coding techniques are used in the streaming session, so that a peer can serve any of its neighbors by streaming information at a constant bit rate. It may be necessary to use a small amount of additional control overhead to send the coefficients at the beginning of the linear combination of data packets. We could account for this additional overhead by increasing the playback rate slightly.

The dedicated streaming server continuously generates data messages that form the multimedia stream. Whenever possible, each peer receives the coded media stream from (possibly several of) their peers. If the serving peers are unable to jointly send messages at the bit rate p, the remaining unserved rate is served from the dedicated server. Hence, the choice of topology has a significant effect on the bandwidth cost to the server. Furthermore, one peer can only serve another if it has already received the data stream. This means that either the peers must query each other to find out which ones have already received the data stream, or they must know which nodes have packets in some other way with only local information (such as ordering in a tree). Therefore, the manner in which peers are chosen to serve can have a substantial impact on the cost to the server, the unused upload bandwidth at peers, and the overall performance of the peer-to-peer streaming session.

IV. Outreach: Analysis and Centralized Algorithm

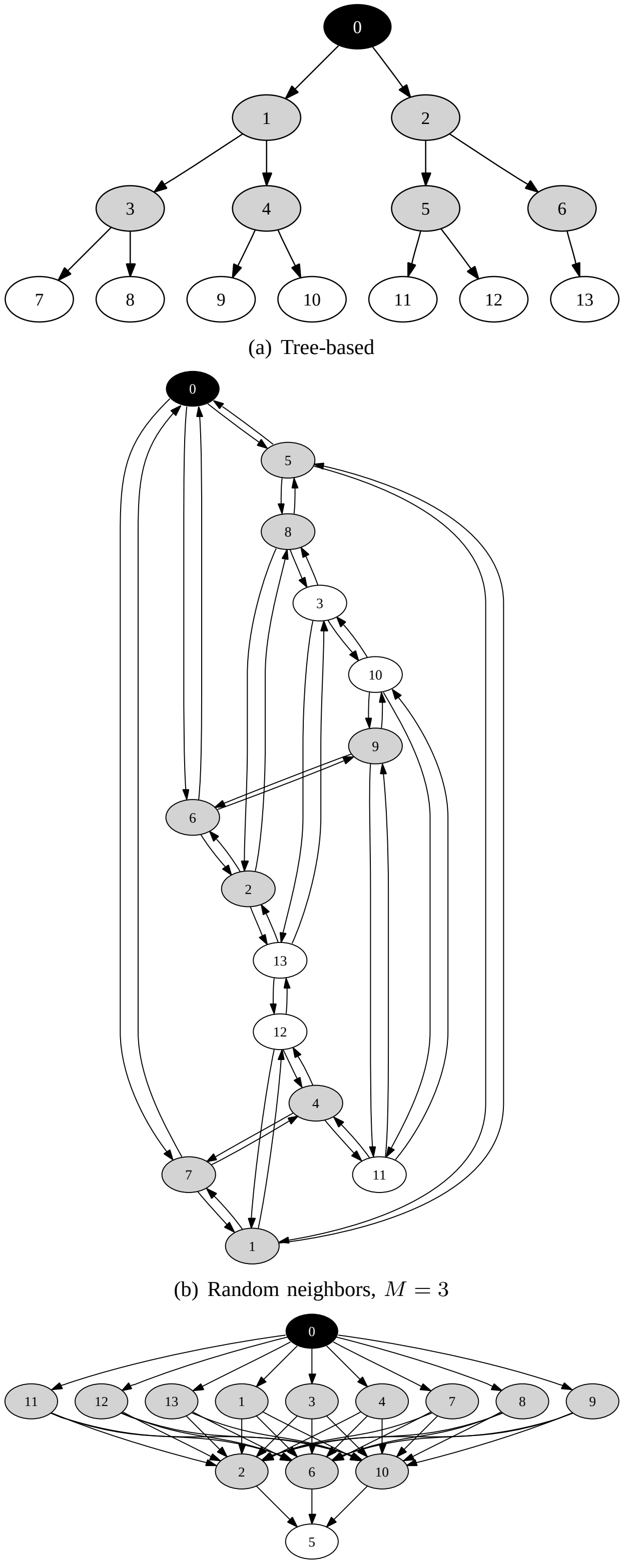
In Outreach, we seek to construct the best possible topologies to minimize server bandwidth costs. To design such high-quality topologies, we first gain insight by studying the characteristics of tree-based and random neighbor topologies that are previously proposed in related work.

A. Tree-Based Topologies

Tree-based peer-to-peer topology constructions arrange participating peers as a tree, by running centralized algorithms at the streaming server to “place” every peer in the topology. A bandwidth-optimized tree constrains the out degree of a peer by its uplink bandwidth. If $U < p$, this implies that the tree can only be organized as a set of *chains* of peers[1] and no parent is able to serve its child completely. The number of chains in the topology depends on the server bandwidth. If the last peer in the chain has the lowest upload bandwidth, this topology would be optimal with respect to the bandwidth cost at the server. The disadvantage of a chain topology is that the delay very quickly becomes intolerably long.

Recall that mesh-based topologies, such as *Bullet*, use underlying trees as the basis to distribute data, and then each peer in the mesh requests the messages that it has not yet received from its neighbors. Ideally, such mesh-based topologies would find the peers with available uplink bandwidth (*i.e.*, the leaves of the trees) to serve the remaining messages; however, finding these peers requires additional control traffic overhead and additional delay. It would be preferable to design an underlying topological structure that did not need such additional overhead.

Fig. 1(a) shows an example of a binary tree topology. Peer 0 is the multimedia server, which serves all the peers in the session to guarantee their playback bit rate. The shaded peers are those that receive the media in time to relay the data to other peers that have not yet received it. The unshaded peers are those that cannot use their upload bandwidth to help serve because all peers have already received the data. Clearly, there are several peers in the binary tree topology that cannot use their upload bandwidth.



(a) Tree-based

(b) Random neighbors, $M = 3$

(c) Outreach, $U = u$, $p = 3u$

Fig. 1. Possible peer-to-peer topologies with $N = 14$.

[1] If only one peer connects directly to the server and every peer has exactly one child, we call this type of tree topology a *chain*.

B. Topologies Involving Random Neighbors

Another possible approach of arranging peers in a topology requires each peer, including the server, to choose M random peers as “partners” that could potentially serve the media stream. The structure of such a topology is unaffected by different peer uplink capacities and different link latencies between peers. When new peers are added to the topology, they search for M neighbors and are connected to the server (in one large connected component) with high probability; however, the topology is not optimized with respect to any particular metric or objective.

Fig. 1(b) shows an example of the random neighbor topology with $M = 3$. Compared to a binary tree topology, we observe that more peers are able to assist the server and use their upload bandwidth to serve other peers (represented by shaded circles), but there are still several peers with unused uplink bandwidth (represented by unshaded circles).

C. The Outreach Topologies

With escalating server bandwidth costs when higher-bandwidth uplink are deployed, we seek to take advantage of the upload bandwidth of participating peers as much as possible. Unlike tree-based topologies, *Outreach* constructs optimized topologies that not only minimize the server bandwidth cost, but also achieves reasonable delays by allowing a peer to be served by multiple other peers in the session.

We construct the Outreach topology of N peers “from the ground up,” in the sense that we first consider peer l, the peer that receives the data stream at *the latest* time. Peer l could not forward any portions of the stream to any other peer because all other peers would already have received it; therefore, the upload bandwidth of peer l must remain idle. We would achieve the best result if peer l had the least upload bandwidth, and peer l was the only peer to receive the stream last (allowing all other peers to assist in serving). In practice, when we introduce peer dynamics and churn into the peer-to-peer session, it may not be beneficial to force the peer with the smallest upload bandwidth as peer l at all times, as this would be highly disruptive in a highly dynamic peer-to-peer environment. For this reason, there is only one peer who is the latest to receive the stream in Outreach topologies, though it may not always have the lowest upload bandwidth of any peer.

We next consider the peers serving peer l. We choose a set of peers from the remaining $N - 2$ such that the combined upload capacity of the peers is as close as possible to p (but not greater). These peers are referred to as the $(l-1)$-peers. In a similar fashion, we assign a collection of peers to serve each of the $(l - 1)$-peers; however, we allow some of the excess peer upload bandwidth to serve the small unserved portion of peer l. This process is repeated until all peers are filled into positions in the topology, and any remaining peer that is not served by other peers is served by the server. We can refer to the set of peers served by the server as 1-peers, the peers they serve as 2-peers, and so on.

The server bandwidth cost in a peer-to-peer topology is the difference between the bandwidth required by the peers to stream the data messages and the uplink bandwidth contributed by the peers. Since, by construction, this topology uses the most possible upload bandwidth from the peers, it imposes minimial server cost. Every peer except peer l is using its entire upload bandwidth to serve other peers.

Fig. 1(c) shows an example of the Outreach topology where all peers have identical upload bandwidth $U = p/3$. From the comparison in Fig. 1, we see that Outreach has the smallest number of peers with idle uplink bandwidth, and therefore most efficiently utilizes server bandwidth.

Clearly, this topology cannot be implemented by reconstructing the network as described above each time a peer arrives or departs. Luckily, the Outreach topology can be approximated by inserting peers using the greedy Algorithm 1, with no topology reconstruction upon peer departure.

D. Outreach: Centralized Algorithm

Algorithm 1. Immediately before a new peer joins the streaming session, let the bandwidth difference between the k- and $(k + 1)$-peers be

$$\delta_k := \begin{cases} \left(\sum_{j\in\mathscr{P}_k} e_j\right) - \left(|\mathscr{P}_{k+1}| * p - \sum_{m\in\mathscr{P}_{k+1}} s_m\right) \\ \qquad\qquad\qquad\qquad\qquad \text{for } 1 \le k < l \\ \sum_{j\in\mathscr{P}_l} u_j \qquad\qquad\qquad\quad \text{for } k = l, \end{cases}$$

where p is the playback rate of the media, u_j is the total upload bandwidth of peer j, e_j is the unused (excess) upload bandwidth of peer j, s_m is the bandwidth served to peer m from other peers, and $\mathscr{P}_i$ is the set of i-peers. Let $k^* = \max_k \arg\{|\delta_k|\}$. If $\delta_{k^*} < 0$, the new peer is inserted as a k^*-peer, which can potentially be served by any (k^*-1)-peer and to help serve any $(k^* + 1)$-peers. If $\delta_{k^*} > 0$, the new peer is inserted as a $(k^* + 1)$-peer, to be served by the k^*-peers and to serve the $(k^* + 2)$-peers. Note that $|\delta_{k^*}| > 0$ because there is necessarily idle upload bandwidth available at the l-peers (i.e., there exists k such that $|\delta_k| > 0$).

When placing peers in the network according to Algorithm 1, we consider an ideal peer-to-peer network environment where:

1) The incoming bandwidth at a peer does not affect its ability to upload to other peers since the server supplies any missing bandwidth to receive the data stream successfully;
2) Any k-peer can serve any $(k+1)$-peer and the bandwidth is infinitely divisible, that is, no assumption is made for minimum packet size.

Lemma 1: Algorithm 1 places peers in a directed graph such that the server cost is increased as little as possible each time a new peer is inserted.

Proof: First note that δ_k of Algorithm 1 is the sum of two parts: the total unused upload from the k-peers and the total amount left to be served at the $(k+1)$-peers. Only one of these two parts will be non-zero. If they were both greater than zero, then the k-peers with idle upload bandwidth would serve the $(k + 1)$-peers that were not completely served. Therefore, if $\delta_k < 0$, there is no idle upload bandwidth from any of the k-peers, but there are some $(k+1)$-peers that are not completely served. If $\delta_k > 0$ then there is available upload bandwidth at some k-peers, so a new peer would be partially served by those peers if it became a $(k+1)$-peer. Note that if peers are added as described above, $\delta_k > 0$ implies that a peer which was previously part of the network failed or went offline.

The additional server bandwidth cost ΔC of adding a new peer i is

$$\Delta C = p - (u_i - e_i) - s_i = p - u_i + e_i - s_i.$$

Peer i may add to the server cost because it requires reception of the media stream at rate p and contributes only u_i; however this cost can be lessened by its choice of insertion position. Peer i can be placed after a peer a that has some idle bandwidth, and can place itself ahead of a peer b that needs to be served (and serve peer b). By finding k^* and δ_{k^*} from Algorithm 1, we select the location that maximizes the sum of the upload bandwidth to the other peers from peer i, and the upload bandwidth from other peers to serve peer i (*i.e.*, it maximizes $[u_i - e_i] + s_i$). Since p is constant, ΔC is minimized. ■

Theorem 1: Inserting each peer according to the locally-optimal Algorithm 1 either creates an Outreach topology from the peers in the network or the network disappears.

Proof: By Lemma 1, we know that Algorithm 1 is locally optimal because it minimizes the increase in server cost at each step; this does not necessarily assure global optimality. However, since our topology allows any k-peer to serve any $(k+1)$-peer, all idle bandwidth from the peers can be used and will be considered for every new peer insertion. This means that the server cost is not only minimized in a local sense, but also in a global sense because the placement of a peer at an early stage will never hamper peer placements at later stages. Furthermore, Algorithm 1 minimizes the unserved bandwidth[2] for the k-peers by definition of δ_k. Therefore, Algorithm 1 constructs an Outreach topology.

If the rate of peer departure is greater than the rate of insertion of new peers, we move toward Outreach each time a peer is inserted, but eventually the number of peers goes to zero and we say that the network disappears.

For a more detailed constructive proof, the interested reader is referred to the Appendix. ■

V. Outreach: Distributed Algorithm

A scalable algorithm cannot be centralized, so we implement Algorithm 1 in a distributed fashion. We assume that the server is too busy to assign positions to all peers entering the system, so the incoming peers are forwarded randomly to *deputies* (1-peers). These deputy peers do not have complete network knowledge, but they retain a small cache of peers that were previously placed in the network and can communicate with the cached peers to learn about the network. This creates two manners by which the distributed algorithm can acquire different levels of knowledge about the system: the number of peers in its cache, and the accuracy of the bandwidth difference (δ_k) estimates of the cached peers, where k is the distance for the cached peer. For simplicity, we assume that the size of the cache remains constant and vary the accuracy of the δ_k estimates. Deputy caches are continually updated to include the most recently inserted peers.

Each cached peer c estimates a δ_k value based on its knowledge of the network. Let w represent a randomly-chosen fraction of the peers for which peer c has approximate knowledge of their uplink and downlink bandwidth usage. As w increases, peer c has more knowledge of the network and can give a more precise estimate of δ_k. For example, if $w = 0.1$, this represents the physical scenario where peer c receives beacons from 10% of the peers, and can communicate with them to send or receive data.[3] This method is designed to recover peers that become disconnected from the connected component of the server; though, since peers express preferential connectivity toward high-degree peers, the network will remain connected with high probability [17]. Algorithm 2 adds peers to the network in a distributed manner.

Algorithm 2. (Distributed version of Algorithm 1.)

Step 1: A new peer i contacts the server to request insertion into the streaming session.

Step 2: Peer i is assigned a deputy (1-peer) for placement.

Step 3: The assigned deputy queries all cached peers for estimates of the bandwidth difference δ_k at their locations in the network, from which it chooses the peer p_{max} which reported the largest $|\delta_k|$.

Step 4: Peer i is assigned a distance d_i, where

$$d_i = \left[\begin{array}{ll} d_{(\text{peer serving } p_{max})} + d, & \text{if } \delta_{k^*} < 0 \\ d_{p_{max}} + d, & \text{if } \delta_{k^*} \geq 0 \end{array} \right.$$

and distance d is chosen from some link delay distribution.

Step 5: Peer i connects with other peers in its neighborhood to serve them, to be served by them, or to keep as "similar peers" to assist in future δ_k estimates by using the following rules:

For each peer j with distance d_j and link l_{ij} between peer i and peer j, and assuming the average link delay is $E(L)$,

- Peer j forms a link to serve peer i, with probability w, if $(d_j + l_{ji} < d_i)$ and $(d_j + l_{ji} > d_i - \frac{1}{2}\mathrm{E}(L))$;
- Peer j forms a link to be served by peer i, with probability w, if $(d_i + l_{ij} < d_j)$ and $(d_i + l_{ij} > d_j - \frac{1}{2}\mathrm{E}(L))$;
- Peer j forms a link to communicate with the new peer as a "similar peer," with probability w, if $[(d_j + l_{ji} > d_i)$ and $(d_j < d_i)]$ or $[(d_i + l_{ij} > d_j)$ and $(d_i < d_j)]$.

Let us elaborate briefly on a cached peer's calculation of δ_k in Step 3. The cached peer c adds up the available upload bandwidth for itself and its "similar peers" and subtracts from this value $p * |\text{peers it could serve}| -$ (amount already served at those peers). This is a simple generalization of the formula in Algorithm 1. Also note that if the

[2] This is the bandwidth required to serve the peers that is not provided by other peers.

[3] As an alternative to beacons, RanSub [16] could be used to obtain a random set of partners.

network is small — for example, with less than 20 peers — we assume $w = 1$ in Algorithm 2, so that the network does not partition. Note that the calculations of δ_k impose only a small additional control overhead.

VI. Performance Evaluation

In our performance evaluation using a simulation-based study, we compare the performance of Outreach to the other topologies discussed in Sec. IV in the presence of churn. Churn refers to the removal and insertion of peers, which is nearly always present in a practical peer-to-peer topology and has the potential to occur at a high rate. New peer arrivals follow a Poisson distribution at rate λ, and peers depart once their heavy-tailed (Zipf-distributed)[4] lifetimes expire. In each case, the system begins with 400 peers, then depending on the relative rates of arrival and expiration, the number of peers increases and decreases over the 600 time-steps of the simulation. The connections of points in the curves indicate points adjacent in time in the simulation. When the rate of insertion of nodes in the network is large, we see points closer to the right side of the curves. When the rate of insertion is lower, then more nodes are expiring than are being introduced; so we see the points of the curve moving toward the left.

First, we compare Outreach to the performance of two tree-based topologies. The *first* tree-based topology is bandwidth-optimized, with four peers connected directly to the server and the other peers forming chains down from those four. This is similar to the chain topology described in Sec. IV-A. By constructing four chains of peers from the server instead of only one, the delays in receiving media streams can be reduced by a factor of four, while introducing only a small amount of additional server cost due to idle bandwidth at leaf peers. The *second* tree topology is a binary tree. In this case, each peer monitors its own reception rate and then checks that its parent is experiencing a similar rate, for tree maintenance and repair. If a peer is removed due to churn in this system, all of its descendents become temporarily disconnected from the server. If any peer experiences a significantly different rate from its parent, then the peer assumes that it is the highest parent of a disconnected subtree and requests reattachment from the central server. We consider the best case scenario where the server is always able to reattach the broken subtrees quickly. Note that this is a conservative estimate, assuming central knowledge and ignores the main drawback of the tree infrastructure.

Recall that the Bullet scheme sends disjoint data sets through the networks, then nodes independently locate the rest of the information. We argue that network coding already sends innovative information in its packets alleviating the need to impose rules for sending disjoint information through the distribution tree. Also, the remaining packets do not need to be located because all information can be retrieved from the data pushed through the topology from the source. In this sense, we say that the tree-based topology that underlies Bullet is already being compared in this performance evaluation.

We then consider topologies constructed with random selections of neighbors (serving peers). If a peer a has fewer than M neighbors, it tries to partner with another peer b at random. A neighbor is refused if a peer b already has M neighbors. The peers can receive from a neighbor if that neighbor is closer to the server (determined by means of a Breadth First Search tree). Any time that a peer has fewer than M neighbors (likely due to peer failures), it has 270 tries in the simulation to find new neighbors[5], chosen from the network uniformly at random. This means that almost all peers can find new neighbors in one step, a very conservative estimate for the sake of comparison.

Outreach performs as described by Algorithm 2 of the previous section. If an Outreach peer has no serving peers (because all serving peers failed, went offline or were serving other peers), then that peer is reinserted into the network by a deputy.

A. Outreach Performance Advantage

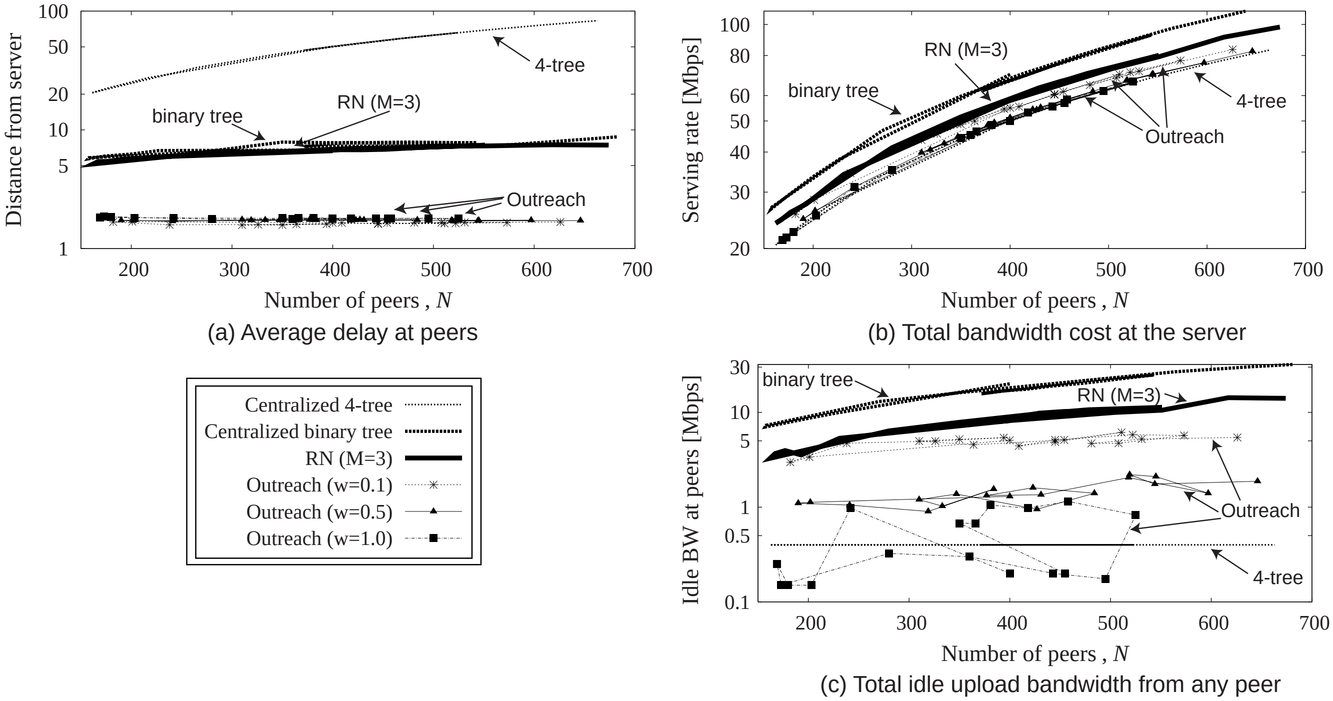
In our first experiment, we compare Outreach with tree-based and random neighbor topologies with respect to the following quality metrics: (1) the distance (latency) from the streaming server; (2) the serving bit rate from the server; and (3) the total idle upload bandwidth at the peers. Our first experiment uses a simplified network where the link lengths are normalized to 1 and the upload capacities are constant. In Fig. 2, each link has play rate $p = 225$ [kbps], and upload rate $U = 100$ [kbps] identical for all peers. The rate of insertion of peers varies throughout the simulation to represent times when there are more and fewer peers online.[6] The insertion process is Poisson, with rate oscillating between 36 peers/min and 1.5 peers/min. The lifetimes of the peers are chosen from a heavy-tailed Zipf distribution, where there is high probability of smaller lifetimes and low probability of long lifetimes. We truncate the Zipf distribution so that the mean is 45 minutes.

In Fig. 2(a), we compare the average distance to the server (the delay of the data messages) as a function of the number of peers, N, in the system. The number of peers varies due to the different rates of insertion and removal. Since Outreach places many peers close to the server, the time from transmission from the server to reception at the peers is very short on average. Normally, one would expect that lowering the delay of packet reception would increase the cost to the media server. Fig. 2(b) shows that the opposite is true, even with conservative assumptions for recovery in the random neighbor and the tree topologies. Outreach achieves lower delay than the other topologies while using less server bandwidth than either the binary tree or the random neighbor topology. The server cost for the centralized tree with four chains achieves cost similar to Outreach. Fig. 2(c) explains this fact. Outreach is specifically designed to use the peer uplink bandwidth and reduce the cost to the server. The 4-chain topology also uses all peer bandwidth except the final four leaves; therefore

[4] The Zipf distribution has been shown to represent the online lifetimes of human users [18].

[5] Such a large number of tries means that there is a very high rate of recovery.

[6] For example, there may be more peers streaming multimedia in the evenings or on the weekends.

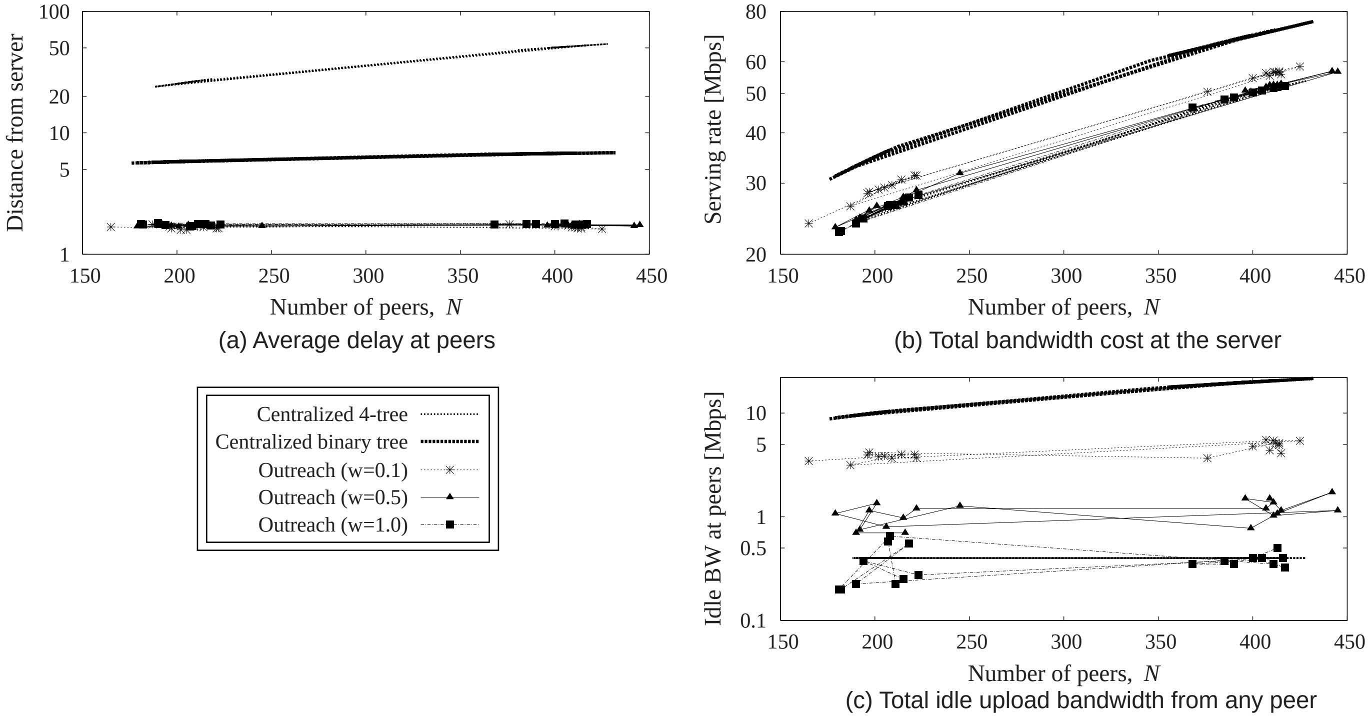


(a) Average delay at peers

(b) Total bandwidth cost at the server

(c) Total idle upload bandwidth from any peer

Fig. 2. Comparison-based studies of Outreach, tree-based and random neighbor topologies (assuming fixed identical upload bandwidth u at all peers, and link delays are all 1).

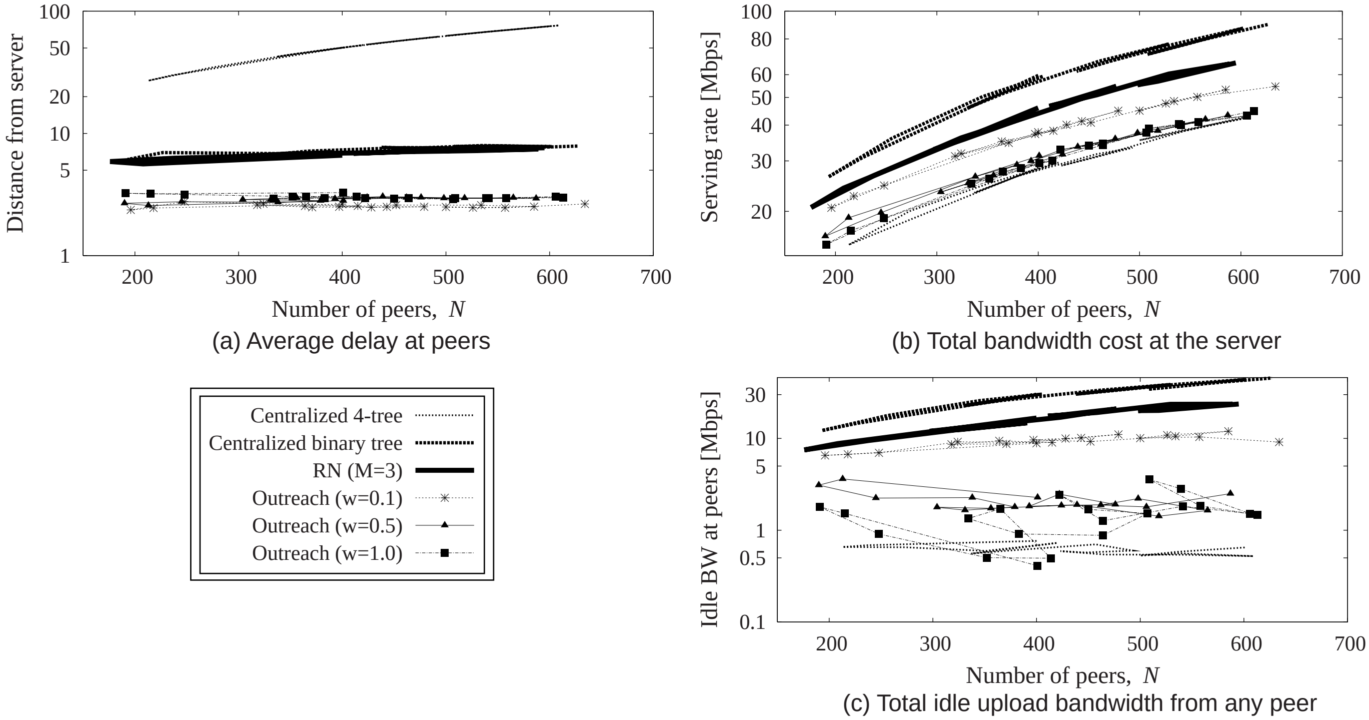


(a) Average delay at peers

(b) Total bandwidth cost at the server

(c) Total idle upload bandwidth from any peer

Fig. 3. The qualities of Outreach, tree-based and random neighbor topologies in a highly dynamic peer-to-peer environment.



(a) Average delay at peers

(b) Total bandwidth cost at the server

(c) Total idle upload bandwidth from any peer

Fig. 4. Outreach, tree-based and random neighbor topologies with peer uplink bandwidth following a Zipf distribution.

achieving similar server costs. Since the other topologies do not optimize for uplink bandwidth utilization at the peers, the server cost is higher.

B. Effects of High Churn Rate

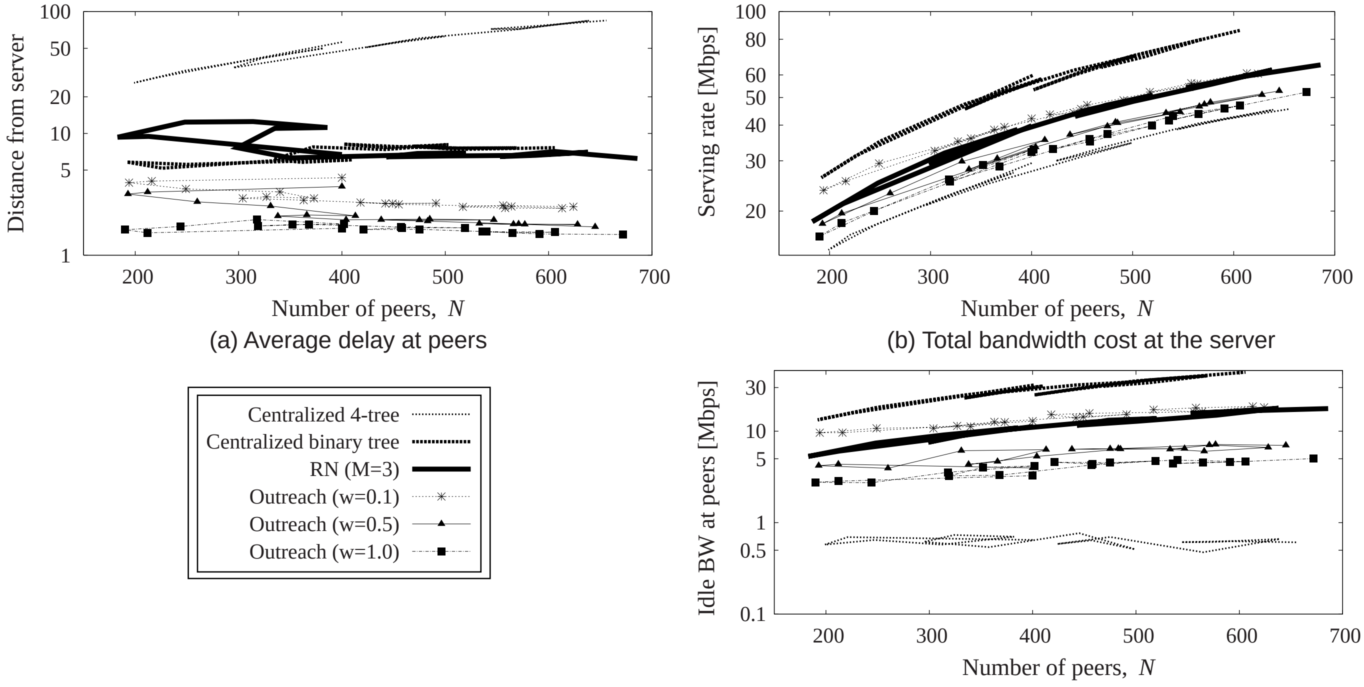
Next, we increase the rate of peer churn, *i.e.*, the number of peers arriving and departing at each time-step. This change has little effect on centralized approaches because they have global knowledge and can adapt quickly, but it can be detrimental to a decentralized approach. However, the graphs in Fig. 3 have demonstrated the fact that, Outreach is extremely resilient to churn. We increase the rate of churn by roughly 100 times by setting the rate of increase λ oscillating between 200 and 400, and by decreasing the average lifetime by 20. We consider the worst case scenario (in terms of variance) at each time-step by first removing all the peers that expire in that time step, then adding all the peers dictated by the Poisson arrival distribution. Note that the server cost depends primarily on the number of peers in the system, and did not perform differently when the peers arrive faster than they depart, for example. The random neighbor case is not shown in these figures because the rate of churn was so high that neighbors could not be found efficiently and peers were almost never connected to the server.

C. Variable Upload Bandwidth and Link Delay

Since it is quite unlikely that all peers will have identical upload bandwidth, Fig. 4 shows the performance of these topologies when the upload capacity is chosen from a Zipf distribution. By using this distribution, we are attempting to take into account that peers have different inherent upload capacities, and that the peers are likely multitasking and using some of their uplink bandwidth for other purposes. As before, the insertion process is Poisson with rate oscillating slowly between 36 peers/min and 1.5 peers/min and the lifetimes are heavy-tailed.

Fig. 4(a) shows that the delays for the tree and random neighbor topologies do not change from Fig. 2. This is consistent because the number and choice of neighbors in these approaches are completely independent from their uplink bandwidth. On the other hand, Outreach does depend on peer uplink bandwidth, so Fig. 4(a) has slightly larger values than Fig. 2(a). If one peer has a particularly high uplink bandwidth, for example rate p, it can serve a peer entirely by itself. As a result, that peer needs fewer similar peers, and new peers are placed at its downstream in the directed graph, so the delays are increased somewhat for Outreach. This increase is not significant, since as the number of peers becomes large, the number of actual i-peers is close to the number of i-peers that would be used in Outreach if each peer had the average uplink bandwidth. We also see from Figures 4(b) and 4(c) that the knowledge fraction, w, is more pronounced. When churn exists in Outreach, the departure of a peer a and the insertion of a new peer b in the same position may have a detrimental effect because the graph was built using uplink information from peer a. In particular, with little knowledge of the system, there is a lower chance that peer b can be placed in a way that compensates for the departure of peer a. In the other topologies, the server cost is increased because the variation in uplink bandwidth is not accounted for, so high-bandwidth peers cannot help the lower-bandwidth peers to serve. Though all topologies perform worse in this case, Outreach improves upon the others.

Finally, the metrics in Fig. 5 represent a network where the



(a) Average delay at peers

(b) Total bandwidth cost at the server

(c) Total idle upload bandwidth from any peer

Fig. 5. Evaluating Outreach, tree-based and random neighbor topologies with uplink bandwidth from a Zipf distribution, and link delays taken from ping data between PlanetLab node pairs.

link delays are chosen from a realistic distribution of ping times between pairs of PlanetLab nodes[7]. Having different delays of links has very little effect on either the tree topology or the random neighbor topology, again because link delays are not considered in the topology design and there is no loss; however, the delays affect the distributed algorithm of Outreach. The main observation here is that there are fewer opportunities for a newly inserted peer to compensate directly for a removed peer, particularly when the knowledge w is reduced.

VII. Conclusion

In this paper, we have presented *Outreach*, a peer-to-peer topology construction algorithm that efficiently uses peer-to-peer uplink bandwidth resources to facilitate scalable multimedia streaming. By introducing a small amount of structure, Outreach optimizes construction of the peer-to-peer overlay with respect to server bandwidth costs, while additionally providing improvement in the delay experienced by the peers. Outreach incorporates a many-to-one service model between peers, since the playback bit rate is often higher than the uplink bandwidth of individual peers. Outreach encourages peers to utilize their uplink bandwidth as completely as possible, while also achieving low latencies.

Our algorithm in Outreach is completely distributed, but is based on an analytical minimization of server cost for high-bandwidth applications in an ideal peer-to-peer network with global knowledge. Since Outreach accommodates more realistic models of uplink capacities and link delays, the Outreach topology achieves better performance than previous topologies in realistic scenarios, and its efficiency allows applications to scale to larger network sizes that may previously have overwhelmed a multimedia source.

In our future work, we plan to implement the Outreach topology construction algorithm in a real-world peer-to-peer streaming application, incorporating advanced coding solutions such as network coding. Once such an implementation is deployed, we plan to collect real-world statistics on its performance in highly dynamic peer-to-peer environments.

Appendix
Proof of Theorem 1

Theorem 1: Inserting each peer according to the locally-optimal Algorithm 1 either creates an Outreach topology from the peers in the network or the network disappears.

Proof: Since the number of peers in the system changes as peers are inserted and removed, average idle upload bandwidth per peer is a fair metric to compare the efficiency of different intermediate topologies as we perform Algorithm 1. We will show that the average idle upload bandwidth $\rightarrow 0$ as the number of peers $\rightarrow \infty$, regardless of the removal strategy of the peers.

First suppose that there are no peer failures, only new peer insertions into an initial directed graph τ. We will show that after some finite time, $|\delta_k| < p, \forall k$. By Algorithm 1, $k^* < l$ (peers are inserted into the middle part of the graph) if $\delta_{k^*} > \sum_{j \in l\text{-peers}} u_j$. Once the bandwidth difference $|\delta_k|$ is less than $\sum_{j \in l\text{-peers}} u_j, \forall k$, an inserted peer will be added to the bottom of the graph to use the upload bandwidth of the l-peers. Then, the number of hops until the last peer receives the

[7] Data taken on December 5, 2005, `http://mybook.uc.edu/ping/`.

message, l, will increase. After some finite time, at each point just before l is increased, it is necessarily true that $|\delta_k| < p$ $\forall k < l$ because the next peer placed at the bottom of the graph would be served at most p and would not serve any other peer.

Suppose we wish to place a new peer i that has upload bandwidth u_i in the network. If peer i is placed as a k^*-peer then we know that there is no idle bandwidth from any k^*-peer (as discussed in the proof of Lemma 1) and that the bandwidth needed to serve the (k^*+1)-peers is at least $\sum_{j \in l\text{-peers}} u_j$. After this insertion, either the (k^*+1)-peers are served completely and there is at most $p - \sum_{j \in l\text{-peers}} u_j$ idle upload at the k^*-peers, or all of the upload capacity of peer i is used. Since k^* maximizes $|\delta_k|$, if any idle bandwidth remains at the k^*-peers means that $\forall k$ $|\delta_k| < p$ before the placement of peer i. However, we have introduced a new k^*-peer, so δ_{k^*-1} is decreased by p. This decrease may cause a ripple effect. If new (k^*-1)-peers are added to compensate for $\delta_{(k^*-1)}$, $\delta_{(k^*-2)}$ will decrease and so on possibly until 1-peers are added. The 1-peers can always be served without introduction of new peers since they are connected directly to a server that serves them completely. After this ripple, $|\delta_k| < p$, $\forall k$. If peer i uses all of its upload bandwidth to serve the (k^*+1)-peers, then $\delta_{k^*} > p$ before the peer insertion. It is possible that $\delta_{k^*} > p$ after the insertion, but δ_{k^*} is necessarily decreased by p. Therefore, this value of k^* could not be chosen indefinitely and the process would have $\delta_{k^*} < p$ in finite time. In this case, a ripple may still be induced from $\delta_{(k^*-1)}$ up to δ_1; however, there it may pause if there are other k such that $|\delta_k| > p$. Luckily, ripples of peer additions do not interfere, so there are no conflicts and the δ_k where $\delta_k > p$ are decreased (allowing for ripples) after each insertion.

Alternatively, if peer i is placed as a (k^*+1)-peer then we know that there is at least $\sum_{j \in l\text{-peers}} u_j$ idle bandwidth from the k^*-peers and that all other (k^*+1)-peers are served completely. After this insertion, either the (k^*+1)-peers including peer i are served completely and the upload at the k^*-peers is reduced by p, or peer i is partially served and there is no remaining upload bandwidth at the k^*-peers. If peer i is only partially served, that means there was less than p unused upload bandwidth at the k^*-peers, and $\forall k$, $|\delta_k| < p$ before the placement of the peer because k^* maximizes $|\delta_k|$. After the peer is placed, we may again need to use the ripple technique, but in the other direction (away from the server). Since we have introduced a new (k^*+1)-peer, $\delta_{(k^*+1)}$ is increased by u_i. If new (k^*+2)-peers are added, $\delta_{(k^*+2)}$ will increase and so on possibly until a new peer is added to the bottom of the graph (after the previous l-peers). After this ripple, $|\delta_k| < p$, $\forall k$. If peer i is served completely then $|\delta_{k^*}| > p$ before the peer insertion. It is possible that $\delta_{k^*} > p$ after the insertion, but $|\delta_{k^*}|$ is necessarily decreased by p so this choice could not be repeated indefinitely and $\delta_{k^*} < p$ in finite time. In this case, a ripple may still be induced from $\delta_{(k^*+1)}$ down to δ_l; however, it may pause if there are other $|\delta_k| > p$. Note that ripples do not conflict with each other. If there is a case where k-peers are added to provide upload bandwidth, then $(k+1)$-peers, then k-peers again to as dictated, in a type of iteration on two adjacent k-values, the process could not iterate forever. In fact, if this process iterated, the $|\delta_{k-1}|$ and $|\delta_{k+1}|$ would quickly grow large creating ripples both from k toward the server and from k toward the bottom of the graph. Since oscillations cannot be created due to the ripples, we say that ripples do not interfere.

Once the discrepancy between upload bandwidth provided by the k-peers and bandwidth used by the $(k+1)$-peers is bounded to p, $\forall k$, the newly inserted peer will be added to the bottom of the graph. This insertion may increase the total idle upload bandwidth of the peers slightly if the upload of peer i is larger that $\sum_{j \in l\text{-peers}} u_j$; however, this increase is bounded by p because upload capacities were defined to have size between 0 and p.

Therefore, the total idle upload bandwidth from the peers is bounded by lp, where l increases with the number of peers in the network. As N becomes large, the number k-peers also becomes large for all k, except k close to l that contain very few peers. We know from the law of large numbers that $U_1 + U_2 + ... + U_n \approx u * n$, for large n and $u = \mathrm{E}(U)$. Therefore the number of peers one hop from the server approaches $(\frac{p}{u})^l$ and the total number of peers approaches $\sum_{i=1}^{l} (\frac{p}{u})^i = \frac{1-(p/u)^{l+1}}{1-(p/u)}$. This means $l \sim O(\log N)$ for a system with N peers, for large N. The average idle upload bandwidth for a peer, $\frac{O(\log N)}{N}, \to 0$ as $N \to \infty$.

Next, consider peers that expire.

If the rate of adding peers to the network is greater than the rate of peer removal $(\lambda > \mu)$, then more peers are introduced into the system. If a peer fails, then one of the new peers can replace it. The remaining new peers follow insertions as described above and the average idle upload bandwidth for a peer $\frac{O(\log N)}{N} \to 0$ as $N \to \infty$.

If $(\lambda < \mu)$, then the network size reduces until there are no more peers, so we say that the network disappears. If $(\lambda = \mu)$, then by Lemma 1 the server cost is minimized as much as possible; however, depending in initial topology and depending on the method by which peers are removed, Algorithm 1 will bring the topology closer to Outreach at each step, but may never reach actually reach an Outreach topology.

Since this topology approaches the minimum server bandwidth and has multiple peers serving other peers at the same time, either it approximates Outreach or there are no more peers in the network because the peers expire more quickly than they are inserted. ■

References

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

[2] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *Proc. 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, May 2002.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth content distribution in a cooperative environment," in *Proc. 2nd International Workshop on Peer-to-Peer Systems*, February 2003.

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. SIGCOMM'02*, August 2002.

[5] D. Tran, K. Hua, and T. Do, "A Peer-to-Peer Architecture for Media Streaming," in *IEEE Journal on Selected Areas in Communications*, vol. 22, January 2004, pp. 121–133.

[6] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," in *Proc. 19th ACM Symposium on Operating Systems Principles*, October 2003.

[7] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient Multicast using Overlays," in *Proc. SIGMETRICS'03*, June 2003.

[8] M. Zhang, L. Zhao, J. L. Y. Tang, and S. Yang, "GridMedia: A Peer-to-Peer Network for Streaming Multicast Through the Internet," in *Proc. ACM Multimedia 2005*, November 2005.

[9] V. Venkatraman and P. Francis, "ChunkySpread Overlay Multicast," in *Proc. 2nd Symposium on Networked Systems Design and Implementation*, May 2005.

[10] B. Cohen, "Incentives Build Robustness in BitTorrent," in *Proc. Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[11] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "PROMISE: peer-to-peer media streaming using CollectCast," in *Proc. ACM Multimedia 2003*, November 2003.

[12] R. Rejaie and A. Ortega, "PALS: Peer-to-Peer Adaptive Layered Streaming," in *Proc. 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, June 2003.

[13] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," in *IEEE Transactions on Information Theory*, vol. 46, no. 4, July 2000, pp. 1204–1216.

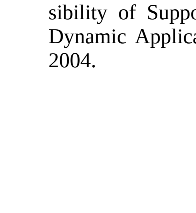
[14] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. IEEE INFOCOM 2005*, April 2005.

[15] J. Widmer and J. L. Boudec, "Network Coding for Efficient Communication in Extreme Networks," in *Proc. WDTN Workshop at SIGCOMM'05*, August 2005.

[16] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat, "Using Random Subsets to Build Scalable Network Services," in *Proc. USENIX Symposium on Internet Technologies and Systems*, March 2003.

[17] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proc. Multimedia Computing and Networking 2002*, January 2002.

[18] 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. SIGCOMM'04*, September 2004.



Tara Small is currently a Postdoctoral Fellow in the Department of Electrical and Computer Engineering at the University of Toronto. Tara studied Mathematics and Physics as an undergraduate at the University of New Brunswick, earning the Governor General's medal for highest standing at the university at her graduation in 2000. In graduate school at Cornell University, Tara concentrated on Applied Mathematics, earning her M.S. in January 2004. With the support of an O'Brien Foundation Fellowship, she completed her Ph.D. dissertation the next year and received her Ph.D. degree in August 2005. She is presently interested in research involving mathematical modeling and optimization of multimedia streaming applications with network coding. Her email address is tsmall@eecg.toronto.edu.

Baochun Li received his B.Engr. degree in 1995 from Department of Computer Science and Technology, Tsinghua University, China, and his M.S. and Ph.D. degrees in 1997 and 2000 from the Department of Computer Science, University of Illinois at Urbana-Champaign. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently an Associate Professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services since October 2003, and the Bell University Laboratories Endowed Chair in Computer Engineering since August 2005. In 2000, he was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems. His research interests include application-level Quality of Service provisioning, wireless and overlay networks. He is a senior member of IEEE, and a member of ACM. His email address is bli@eecg.toronto.edu.

Ben Liang received honors simultaneous B.Sc. (valedictorian) and M.Sc. degrees in electrical engineering from Polytechnic University in Brooklyn, New York, in 1997 and the Ph.D. degree in electrical engineering with computer science minor from Cornell University in Ithaca, New York, in 2001. In the 2001 academic year, he was a visiting lecturer and post-doctoral research associate at Cornell University. He joined the Department of Electrical and Computer Engineering at the University of Toronto as an Assistant Professor in 2002. His current research interests are in mobile networking and multimedia systems. He won an Intel Foundation Graduate Fellowship in 2000 to complete his Ph.D. dissertation and the Best Paper Award at the IFIP Networking conference in 2005. He is a member of Tau Beta Pi, IEEE, and ACM. His email address is liang@comm.utoronto.ca.