Scaling Laws and Tradeoffs in Peer-to-Peer Live Multimedia Streaming

Tara Small, Ben Liang, and Baochun Li
Department of Electrical and Computer Engineering
University of Toronto, Toronto, Ontario, Canada
tsmall@eecg.toronto.edu, liang@comm.toronto.edu, bli@eecg.toronto.edu

ABSTRACT

It is well-known that live multimedia streaming applications operate more efficiently when organized in peer-to-peer (P2P) topologies, since peer upload capacities are utilized to support other peers, and to alleviate the load and operating costs on the streaming servers. To date, there have been a number of existing experimental proposals with respect to how such peer-to-peer topologies are organized to support live streaming sessions. However, most of the existing proposals resort to intuition and heuristics when it comes to the design of such topology construction (*i.e.*, neighbor selection) protocols. In this paper, we investigate the scaling laws of live P2P multimedia streaming, by quantitatively studying the asymptotic effects and tradeoffs among three key parameters in P2P streaming: server bandwidth cost, the maximum number of peers that can be supported, and the maximum number of streaming hops experienced by a peer. To further generalize our studies, we do not make restrictive assumptions in our theoretical analysis of such scaling laws: both peer upload capacities and peer lifetimes in a session may come from arbitrary distributions. With the theoretical insights we have developed, we propose *Affinity*, a simple and realistic heuristic to demonstrate the key benefits of our theoretical analysis in dynamic P2P networks, as compared to the topology construction algorithms in existing work.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Distributed Systems—*Distributed applications*; H.3.5 [**Information Systems**]: On-line Information Services—*Data sharing*

General Terms

Algorithms, Design, Theory

Keywords

Peer-to-peer, resource-performance tradeoff, scaling laws, multimedia streaming

1. INTRODUCTION

With the advent of commercial live multimedia streaming services from major ISPs and content providers, research towards live multimedia streaming has never been more promising and realistic. It has been well-known in the community that peer-to-peer (P2P) multimedia streaming from one or multiple multimedia sources (*i.e.*, streaming servers) to a group of participating peers will help to relieve the bandwidth cost burden to the server. If the individual peers contribute as much bandwidth as they consume, the streaming session is scalable to a large number of peers in the session. This observation is intuitive: without utilizing uploading capacities on each peer to assist the downloading demand of other peers, the load and bandwidth cost on the multimedia sources may escalate beyond the operating capacities of streaming servers.

To date, we have observed a number of existing proposals towards the construction of P2P topologies in live P2P streaming sessions (also referred to as *neighbor selection* in the literature). However, they resort to intuition and heuristics when designing such topology construction protocols, by using a tree-based, a randomized, or a directed acyclic graph (DAG) strategy. Without a doubt, devising protocols to construct a "good" or high-quality peer-to-peer topology is critical towards the scalability and robustness of a live P2P streaming session.

What, then, constitutes a P2P topology of high quality? Beyond the basic requirement that the streaming bit rate be sustained on each peer during the session, we are concerned with three important metrics that collectively define the performance and cost of a streaming session: (1) *delay:* the difference between the playback time and the time when live events occur in the original media stream (*i.e.*, the number of hops that the streaming playback must travel to propagate to all network peers); (2) *server cost*: the bandwidth cost on streaming servers; and (3) *scalability*: the maximum number of peers that a session can support. One prefers P2P streaming topologies that scale to a large number of peers, minimizes server costs, and with a bounded delay on each of the peers.

At one extreme of the complete spectrum of topology construction, we consider the case where the multimedia source serves all peers directly. This leads to almost immediate media streaming at all peers (one hop delay for the single direct server connection), but incurs tremendous bandwidth costs at the server. Such unbounded server cost implies that the system is not scalable. At the other extreme, the source could serve data to one peer only. This bounds the server cost, but that one peer then needs to relay the data to other peers, which in turn serves additional peers. In many cases, the peers would not be able to playback the stream at the required streaming rate; and in all cases the delay will be unbounded as the number of peers scales up.

Towards the construction of high-quality P2P streaming topologies, it is natural to ask the following fundamental questions: (1) Given a certain bandwidth cost when operating streaming servers that serve as multimedia sources, what is the maximum number of peers that can be supported in the session? (2) If the peers can tolerate longer delay from the time that the session is first emitted from the source (allowing more forwarding between peers), how much can the server bandwidth cost be reduced? (3) Given a

MM'06, October 23–27, 2006, Santa Barbara, California, USA.

certain bandwidth cost for the streaming servers in a dynamic environment with peer churn (dynamic departures and arrivals), what is the probability that a peer wishing to enter the network would be denied service? Last but not the least, can existing protocols be further improved with answers to these questions, or do they represent the best we can do?

Unfortunately, existing literature on live peer-to-peer streaming fails to offer a comprehensive and analytical study on the characteristics that govern the scalability and performance of live P2P streaming topologies. In this paper, we seek to analytically study the intricate dependencies among the *scalability* of the streaming session, the *bandwidth cost* on streaming servers, as well as the maximum permitted *delays* at the peers. In particular, we theoretically derive overlay formation that achieves the best performance for DAG networks whose routing is not limited by the content delivery mechanism. For example in a P2P system with network coding, it has been shown that nearly every coded block forwarded by a peer is innovative, so locations of particular data blocks do not limit the network performance and all available upload bandwidth can be used at the peers.

The original contributions of this paper are three-fold. *First,* we systematically present the theoretical analysis of the scaling laws of live P2P streaming in the context of directed acyclic graphs, by quantitatively studying the asymptotic effects and tradeoffs of scalability, server cost and delay. *Second,* we consider the dynamic nature of peers in P2P DAG topologies, and propose an analytical framework to estimate its effect on the peer service probability, service waiting time, and disconnection period. *Finally,* while this paper is largely theoretical, we are convinced that our theoretical insights have important practical implications. We introduce *Affinity,* a simple and realistic heuristic that can be readily implemented. We then evaluate Affinity in the context of previously proposed tree-based or randomized topology construction algorithms, with the intent to show a proof-of-concept on how insights on scaling laws can be used in the process of designing practical protocols. Throughout our analysis, we do not make unreasonable or restrictive assumptions: both peer upload capacities and peer lifetimes in a session may assume arbitrary distributions.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 studies the scaling laws of live peer-to-peer streaming in DAG topologies. The effects of peer dynamics are analyzed in Section 4. The *Affinity* heuristic is presented in Section 5, and evaluated in the context of previously proposed strategies. Section 6 concludes the paper.

2. RELATED WORK

Many existing work on constructing peer-to-peer topologies (*i.e.*, neighbor selection) generally fall into two categories: *tree-based strategies*, where each individual data block traverses one or multiple trees, or *randomized strategies*, where there is no global structure and neighbors are determined dynamically on the fly. Mesh-based structures are based on trees, but add bidirectional links between peers to allow for more communication and greater flexibility; however, the locally-defined decisions of the meshes may lead to suboptimal performance. Most recently, directed acyclic graphs have imposed a partial-ordering on the network and show high resilience to failure, overcoming previous performance issues. The peers in DAGs receive from multiple parents and improve graph connectivity.

Early tree-based topologies, such as Overcast [1], form a single-source multicast network that maintains global status at the root of a changing distribution tree to efficiently adapt data distribution to changing network conditions. Overcast leverages some caching and server replication strategies, but is designed as an overlay network that can be incrementally deployed. These networks typically scale to tens or perhaps hundreds of peers. Tree topologies are unfair because the leaf peers will not serve any other peers at all while others may serve many times the stream volume. With a fairness goal that each peer should transmit the same volume that it receives, multipath tree protocols were developed. Splitstream [2] and Bullet [3] use multiple trees to form a mesh, so that peers will be high in one tree for one substream, but lower in the tree for another, thereby achieving fairness and more resilience under churn.

More recent works propose neighbor selection strategies that are more randomized in nature, with the hope of being more scalable. Chainsaw [4] and CoolStreaming [5] are excellent examples of heuristics in this category. Using either of these protocols, each peer only maintains state for a small set of neighbors, which are chosen at random (no explicit topology), and neighbors request information from each other. They also assume that peers are capable of uploading at the desired streaming rate, though this is not necessarily the case. Gridmedia [6] extends this work to achieve higher throughput rates by allowing peers to also "push" data blocks (sending unsolicited blocks without being explicit requested) once the local peer-to-peer topology is established.

More randomized tree-based strategies have been revisited by Chunkyspread [7], which utilizes multiple trees to spread different slices of the media stream. By designing parent selection and loop avoidance algorithm, Chunkyspread constructs non-optimal trees that iteratively improve themselves over time. Parents are swapped using an algorithm called Swaplinks [8] that uses weighted random walks to build random graphs, so that the parent-child relationship are optimized with respect to load and relative latency. Rodriguez *et al.* [9] seek to build and maintain topologies that optimize one or several metrics at the same time, such as delay and cost. Since this is an NP-hard problem, the constraints are typically ordered according to their importance to the user.

Dagster [10] and DagStream [11] are examples of DAG protocols. Dagster uses an incentive-based mechanism to encourage peers to contribute their upload bandwidth to the network by offering lower service rejection probabilities and lower disruption. Dagstream concentrates on locality awareness and connectivity to prevent failure under churn.

Towards building large-scale peer-to-peer streaming sessions, none of the existing work on live peer-to-peer streaming offers a comprehensive study on the asymptotic and tradeoffs among key elements of scalability, performance and cost of topology construction. To the best of our knowledge, this paper represents the first attempt to theoretically investigate the scaling laws of constructing peer-to-peer topologies for live multimedia streaming sessions.

3. SCALING LAWS OF LIVE P2P STREAMING

Toward optimal P2P live streaming, we present a theoretical analysis of the scaling laws for server cost, playback delay, and the number of served peers. We first observe the optimal properties that are intrinsic to the construction of an optimal P2P streaming session. We then analyze the asymptotic effects and tradeoffs of scalability, server cost, and delay.

3.1 Notations and Preliminaries

Let N_s represent the number of peers in a peer-to-peer session. Between each pair of peers (i, j) is a link l_{ij} that has capacity c_{ij}. One of the peers is the multimedia server and the other $N_s - 1$ are peers that receive the media, to be played at some playback rate p Kbps. In this session, every peer has the ability to function both as a media receiver and also as a sender that contributes its uplink bandwidth to relieve the burden that would otherwise be imposed on the multimedia server. We define the server cost C_s as the total bandwidth provided to any peer by the multimedia source. We further denote the propagation delay by d_p and the queueing delay by d_q.

There are no restrictions on the number of *parents* that serve a given peer, or the number of *children* that are served by that peer. However, each peer a_i has a maximum upload capacity of $u(a_i)$ Kbps, from a distribution $F(U)$ with mean $\overline{U}$.[1] Peers are expected to upload with their entire upload capacity u, but they are able to divide that bandwidth among as many children as they wish, in any proportion.

We assume that each peer uploads with its maximum upload bandwidth u at a constant bit rate.[2] In an ideal system, there is no variance in available bandwidth from the peers and that the capacity c_{ij} of link l_{ij} is sufficient to support any upload bandwidth that the source or the peers provide. We further assume that the transmissions are optimally scheduled so that all data blocks are successfully received.

3.2 Characteristics of Optimal Peer-to-Peer Streaming

In this section, we present in several propositions the intrinsic characteristics of an optimal P2P streaming session. We define an optimal session as one that achieves minimum server cost per peer given that all peers receive the media at rate p within a certain maximum tolerated delay without considering fairness to peers. We first show in Proposition 1 that there is no queueing delay in an optimal session. This has two implications. First, there is no need to maintain a data buffer for optimal P2P streaming, and second, given C_s and N_s, an optimal session achieves minimum average *propagation* delay over all peers.

PROPOSITION 1. *$d_q = 0$ for an optimal session where play length is unlimited.*

PROOF. Queueing delays are incurred when data blocks build up in the buffers of peers waiting to be relayed to another peer or waiting to be played at that peer. If all peers are served at rate p and packets can be routed optimally, then the buffers are not needed. The peers would immediately play back the data they receive. On the other hand, if any peers were served at an average rate less than p, then those peers would use some of the buffered data blocks whenever they could not fulfill the playback rate based on their received data. After some time, the peer's buffer would be empty and the peer would experience a disruption in service until sufficient data was buffered. This means that in the optimal case we need only consider $d_q = 0$. □

Consider the following model for an optimal streaming session of N_s peers with C_s bandwidth provided by the source. We label the N_s peers $a_1, a_2, ..., a_{N_s}$. We will attempt to serve the peers in that order. Let $d(a_i)$ be the propagation delay from the source to peer a_i, that is, the number of hops. Let $u(a_i)$ be the upload capacity of peer a_i. A peer is served if the upload capacity of the source and all of the existing served peers have sufficient bandwidth to serve it at rate p while continuing to serve all previously assigned peers. More formally, $d(a_j) = k+1$ if $k \in \{0, 1, 2, ...\}$ is the smallest value for which $\sum_{i=1}^{j-1} u(a_i)\ \ \{d(a_i) \leq k\} + C_s \geq j \cdot p$, and is the indicator function, i.e. $(x) = 1$ if x is true, and $(x) = 0$ otherwise. If the sum of upload capacities for peer a_i at level k and below and the server's capacity is less then the required total rate $j \cdot p$, then the new peer a_j must be at level $k+1$. Then, the average propagation delay over all peers is $\frac{1}{N_s}\sum_{i=1}^{N_s} d(a_i)$.

Our goal in the optimal streaming session is to minimize the cost to the server for a given delay constraint; that is, we reduce the cost to the server per peer as much as possible and assure that the maximum delay of any peer is not more than some maximum delay d_p. To accomplish this, we maximize the total upload bandwidth contribution by the peers in the network. If there is a large peer bandwidth contribution being utilized by other peers, then the delay of each peer is low. In fact, maximizing the peer bandwidth contribution is equivalent to minimizing the delay of each peer, or equivalently minimizing the sum of all the propagation delays.

The next proposition provides a general guideline on how to construct an optimal P2P streaming topology to minimize the average propagation delay based on the distribution of peer upload capacities (which is the same as minimizing the sum). It is presented as a lemma, to be used in proving the main theorem on scaling laws and performance tradeoff in Section 3.3.

LEMMA 1. *In an optimal session with C_s bandwidth available from the source and peers with upload capacities following a distribution $F(U)$, placing peers with higher upload capacity closer to the source achieves optimal performance for the session (i.e. minimum average delay for all peers).*

PROOF. (by contradiction) Using the model above, consider a session with N_s peers each having $i.i.d.$ upload capacity taken from the probability distribution $F(U)$. Suppose that the peers are ordered to make an optimal topology T (with minimal average delay); however, for at least one pair of peers (a_{n_1}, a_{n_2}), the upload capacity of a_{n_1}, $u(a_{n_1})$, is less than the upload capacity of a_{n_2}, $u(a_{n_2})$, but $d(a_{n_1}) < d(a_{n_2})$. The average delay for peers in this session is

$$D(T) = \frac{1}{N_s}\sum_{i=1}^{N_s} d(a_i) = \frac{1}{N_s}\left\{\sum_{i=1}^{a_{n_1}-1} d(a_i) + d(a_{n_1}) + \sum_{i=a_{n_1}+1}^{a_{n_2}-1} d(a_i) + d(a_{n_2}) + \sum_{i=a_{n_2}+1}^{N_s} d(a_i)\right\} \quad (1)$$

Form an alternate topology T' that is identical to T, but swap the positions of peers a_{n_1} and a_{n_2} so that a'_{n_1} is in the position of a_{n_2} and a'_{n_2} is in the position of a_{n_1}. Then if we let $d'(a_i)$ represent the distance from peer i to the source,

$$D(T') = \frac{1}{N_s}\sum_{i=1}^{N_s} d'(a'_i) = \frac{1}{N_s}\left\{\sum_{i=1}^{a_{n_1}-1} d(a_i) + d'(a'_{n_2}) + \sum_{i=a'_{n_1}+1}^{a'_{n_2}-1} d'(a'_i) + d(a_{n_1}) + \sum_{i=a'_{n_2}+1}^{N_s} d'(a'_i)\right\} \quad (2)$$

we can compare the difference between the average packet delays for the two topologies

$$N_s[D(T) - D(T')] = \sum_{i=a_{n_1}+1}^{a_{n_2}} d(a_i) + d(a_{n_2}) + \sum_{i=a_{n_2}}^{N_s} d(a_i) - d'(a'_{n_2}) - \sum_{i=a'_{n_1}+1}^{a'_{n_2}} d'(a'_i) - \sum_{i=a'_{n_2}}^{N_s} d'(a'_i). \quad (3)$$

Recall that $d(a_j) = k+1$ if k is the smallest value for which $\sum_{i=1}^{j-1} u(a_i)\ \ \{d(a_i) \leq k\} + C_s \geq j \cdot p$. Since $u(a'_{n_1}) > u(a_{n_1})$ and $u(a'_j) = u(a_j), \forall j, j \neq n_1, j \neq n_2$ by assumption, that means $\sum_{i=1}^{j} u(a'_i) > \sum_{i=1}^{j} u(a_i)$ and

$$d(a'_j) \leq d(a_j) \text{ for any } j, n_1 < j \leq n_2. \quad (4)$$

We also know by assumption that $u(a'_{n_2}) < u(a_{n_2})$, but also $u(a'_{n_1}) + u(a'_{n_2}) = u(a_{n_1}) + u(a_{n_2})$. This implies that $\sum_{i=1}^{j} u(a'_i) =$

[1] We assume that P2P traffic has precedence over all other traffic, so all upload capacity u can be utilized.

[2] Clearly this is not realistic, and by adding small buffers at the peers, we will be able to account for variance in the transmission of streaming media from peers; however, we are first determining bounds using an optimal scenario with perfect scheduling and link usage.

$\sum_{i=1}^{j} u(a_i)$ for $j > n_2$. Equation (4) holds, however, and we realize that $\{d(a_i') \leq k\}$ is a superset of $\{d(a_i) \leq k\}$, so

$$d(a_j') \leq d(a_j) \text{ for } j > n_2. \quad (5)$$

With the relationship from (4) and (5), we can see that (3) gives $N_s[D(T) - D(T')] > 0$, contradicting the optimality of T. □

Lemma 1 immediately implies that, in an optimal P2P streaming session, no peer more than one hop away from the server should receive any bandwidth from the server to make up for bandwidth deficiency from its parents. This is stated as Corollary 1.

COROLLARY 1. *In an optimal session (a network with minimum server cost) with C_s bandwidth available from the source, the server will never contribute bandwidth to any peer a_j with $d(a_j) \geq 2$.*

PROOF. Saving some of the server bandwidth C_{save} for peers with some later delay, say m, is equivalent to having a server with bandwidth $C_s - C_{\text{save}}$ and two peers a_s and a_m in the session where a_s has 0 upload bandwidth, but is supposed to serve the peers 1 hop from the server and a_m has C_{save} bandwidth but is placed m hops away. In our notation, this means that a peer with lower bandwidth is placed before a peer with higher upload bandwidth, so swapping the positions of a_s and a_m leads to better performance. That is, the server should use all of its bandwidth to serve the peers completely and not fill in bandwidth to peers that are being served by other peers. □

Finally, to provide a worse-case performance bound, Lemma 2 states that the largest average delay is incurred if all peers have the same uploading capacity. Hence, we obtain a surprising conclusion that, the heterogeneity in peer-device capabilities and network connection quality, which is an prominent characteristic of today's Internet, actually *improves P2P streaming performance*!

LEMMA 2. *The constant upload capacity distribution $F(U) = \overline{U}$ leads to the worst performance (largest average delay) in the optimal session that has mean uploading capacity of $\overline{U}$ per peer.*

PROOF. Using the same model as earlier, let us compare the upload capacity distribution $F(U) = \overline{U}$ to any other distribution of upload capacities $F_{\text{other}}(U)$. As described in Lemma 1, the peers are arranged in decreasing order of upload capacity, so that the peers with higher capacity are placed closer to the source. We can label the peers in these orderings as $a_1, a_2, ...a_{N_s}$, for the peers that all have constant upload capacity, and $b_1, b_2, ...b_{N_s}$, for the peers that all have upload capacity taken from the $F_{\text{other}}(U)$ distribution. Since the upload capacity distributions have the same mean, $\frac{1}{N_s}\sum_{i=1}^{N_s} u(a_i) = \frac{1}{N_s}\sum_{i=1}^{N_s} u(b_i)$.

If $F_{\text{other}}(U)$ has a non-constant distribution, then at least one peer in the b_i list has upload capacity larger than the mean, so $u(b_1) > u(a_1)$. The relative difference of the terms in the sequence of $\sum_{i=1}^{N_s} u(b_i)$ is always non-decreasing for a non-constant upload distribution because the b_i are non-negative. As we can see in Fig. 1, this means that since the two curves must agree at $\frac{1}{N_s}\sum_{i=1}^{N_s} u(a_i) = \frac{1}{N_s}\sum_{i=1}^{N_s} u(b_i)$, they will not cross at any other point and $\frac{1}{N_s}\sum_{i=1}^{N_s} u(a_i) < \frac{1}{N_s}\sum_{i=1}^{N_s} u(b_i)$ at all intermediate i. Equivalently, this implies that $d(a_i) \geq d(b_i)$ so the constant upload capacity distribution experiences the greatest average playback delay. In turn, this leads to the worst performance for the P2P session because the server cost would need to be increased to achieve the same maximum tolerated delay for all peers. □

3.3 Resource-Performance Tradeoff

In this section, we quantify the tradeoff between the server cost C_s, the delay bound d_p, and the maximum number of peers N_s

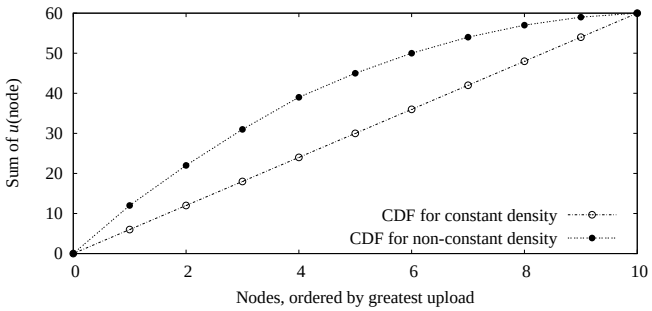

Figure 1: Sum of total upload capacity $\sum_{i=1}^{N_s} u(b_i)$ and $\sum_{i=1}^{N_s} u(b_i)$ as i increases

that can be served. It then culminates to the main conclusion stated in Theorem 1, where we express scaling laws relating these system parameters for optimal streaming.

Suppose that there are N_s peers that wish to be part of the P2P session. Further assume that the peers all have upload distribution $F(U)$. Lemma 1 dictates that we must place the peers with higher upload capacity closer to the source, so as to serve as many peers as possible. Hence, the first $\frac{C_s}{p}$ peers with the highest upload capacities are served by the source. Call them the 1-peers. If we can write the $F(U)$ distribution as a smooth, differentiable function (to be relaxed later) and $f(u) = F'(u)$, then the amount of upload bandwidth available from the 1-peers is

$$A_1 := N_s \int_{F^{-1}(h_1)}^{\infty} uf(u)\mathrm{d}u, \text{ for } h_1 = 1 - \frac{C_s/p}{N_s}. \quad (6)$$

Consider the components of this expression. The $\frac{C_s}{p}$ peers with the highest upload capacity make up the largest $\frac{C_s/p}{N_s}$ fraction of the $F(U)$ distribution curve. This explains why the integral in (6) runs over the interval $(F^{-1}[1 - \frac{C_s/p}{N_s}], \infty)$. The term $N_s f(u)$ corresponds to the number of peers that have upload bandwidth capacity values in the small interval $\mathrm{d}u$ under consideration. Finally, the u factor is the upload capacity that the peers in the $\mathrm{d}u$ interval are able to contribute to the combined upload of the 1-peers.

Note that we have defined two quantities, h_1 and A_1. $h_1 = (1 - \frac{C_s/p}{N_s})$ delimits a boundary because each peer with upload bandwidth of $F^{-1}(h_1)$ or greater is exactly one hop from the multimedia source. A_1 quantifies the amount of capacity available from the 1-peers. Furthermore, we know that $\frac{A_1}{p}$ 2-peers can be supported because they are necessarily served by the collection of 1-peers.

Clearly since we know the number of 2-peers, we can calculate $h_2 = (1 - \frac{C_s/p}{N_s} - \frac{A_1/p}{N_s})$ and $A_2 := N_s \int_{F^{-1}(h_2)}^{F^{-1}(h_1)} uf(u)\mathrm{d}u$. This process is continued to obtain (h_i, A_i), for $i = 2, 3, \ldots$. At some point, this process will terminate for one of three reasons:

1. All N_s peers are used (i.e., $1 > \frac{C_s/p}{N_s} - \frac{A_1/p}{N_s} - \ldots - \frac{A_i/p}{N_s}$),
2. The maximum allowed number of hops, d_p, is reached (before all of the peers have been placed), or
3. There is insufficient peer upload bandwidth from the last group of peers to support any more peers (i.e., $A_i < p$).

Consider these termination conditions separately. If the procedure terminates by condition 1, then we are finished. All peers in the P2P session is partitioned into hops based on their upload capacity. On the other hand, if the procedure terminates by condition 2, this implies that the N_s chosen at the beginning of the procedure was too large. As a result, $N_s f(u)$ would be inflated and we would assume there were more high upload peers than there truly were.

However, this gives us an upper bound on N_s, so we could use the bisection method to find the largest value of N_s that allows the procedure to terminate by condition 1.[3] An alternative is to increase the server capacity C_s to support the N_s peers than are desired. Again, the bisection method can be used to find the corresponding C_s. Similarly, if the procedure terminates by condition 3, then we must iterate the procedure to find the correct N_s.

By defining the h_i values as above, we partition the $F(U)$-axis of the cumulative distribution function of upload capacities. The peers with uploads between $F^{-1}(h_i)$ and $F^{-1}(h_{i-1})$ are positioned i hops from the server. The total upload capacity of the peers i hops from the server, A_i, is used to serve the peers $i+1$ hops from the server.

Recall that even if $F(U)$ has jumps, that is, $F(U)$ has point masses of probability, the cumulative distribution function is defined to be *left-continuous*. Specifically, $F^{-1}(y) = \inf\{s : F(s) \geq y\}$. In this case, we continue to use the h_i variables to partition the probability space to indicate the number of hops from different groups of peers to the source, but the A_i cumulative upload capacities must be defined slightly differently for every interval that includes a point mass.

Without loss of generality, let $[h_j, h_{j-1}]$ be the first upload interval (interval with the largest upload values) that includes a jump in the cumulative distribution function over $[F^{-1}(h_j), F^{-1}(h_{j-1})]$. If $F[F^{-1}(h_j)] = h_j$, then all peers with upload h_j are included in the interval, that is, both ends of the jump are mapped into the interval $[h_j, h_{j-1}]$. Otherwise, some of the peers with upload $F^{-1}(h_j)$ can be served in j hops, but others will be farther from the source because there is insufficient bandwidth to support them at j hops.

When $F[F^{-1}(h_j)] = h_j$, we obtain essentially the same definition of A_j as before, except that the point masses at all non-differentiable points must be included. No other intervals are affected. For a set of non-differentiable points $x_1, x_2, ..., x_m$,

$$A_j := N_s \left(\int_{F^{-1}(h_j)}^{x_1} uf(u)\mathrm{d}u + \ldots + \int_{x_m}^{F^{-1}(h_{j-1})} uf(u)\mathrm{d}u + x_1 P(U = x_1) + ... + x_m P(U = x_m) \right).$$

On the other hand if $F[F^{-1}(h_j)] \neq h_j$ (which necessarily means that $F[F^{-1}(h_j)] < h_j$), then the calculation of A_j and subsequent A_i for $i > j$ are affected. The fraction of peers with upload capacity that can be served j hops from the source is $\frac{r_j - h_j}{r_j - l_j}$, where $r_j = \lim_{x \to h_j} F[F^{-1}(x)]$ is the limit value of the jump from the right side, and $l_j = F[F^{-1}(h_j)]$ is the value of the jump from the left side. Or for a set of non-differentiable points $x_1, x_2, ..., x_m$ where x_1 has value $F^{-1}(h_j)$,

$$A_j := N_s \left(\int_{x_1}^{x_2} uf(u)\mathrm{d}u + ... + \int_{x_m}^{F^{-1}(h_{j-1})} uf(u)\mathrm{d}u + \frac{r_j - h_j}{r_j - l_j} P(U = x_1) + ... + x_m P(U = x_m) \right),$$

for r_j and l_j defined as above. Since $\frac{r_j - h_j}{r_j - l_j}$ of the peers with $F^{-1}(h_j)$ peers are used in A_j, there are still $\frac{h_j - l_j}{r_j - l_j}$ of them available to be placed so they contribute to A_{j+1}. If $F[F^{-1}(h_j)] = F[F^{-1}(h_{j+1})]$, then all peers in the $[h_{j+1}, h_j]$ interval have the same upload capacity and A_{j+1} is simply $N_s \frac{(r_j - h_j) - h_{j+1}}{r_j - l_j} P(U = F^{-1}(h_j))$ and A_{j+2} will also be affected by the jump at $F^{-1}(h_j)$.

[3]In the bisection approach, we first repeat the procedure using $N_s/2$ peers and keep halving the intervals until we find a feasible N_s. We then successively half the interval between a feasible N_s and the most recent over-estimation of N_s, until the interval converges to the maximum feasible N_s.

Otherwise the $[h_{j+1}, h_j]$ interval contains peers with upload capacity less than $F^{-1}(h_j)$ (less than the value of the jump) and for a set of non-differentiable points $x_1, x_2, ..., x_m$ where x_m has value $F^{-1}(h_j)$,

$$A_j := N_s \left(\int_{F^{-1}(h_{j+1})}^{x_2} uf(u)\mathrm{d}u + ... + \int_{x_{m-1}}^{x_m} uf(u)\mathrm{d}u + \frac{r_j - h_j}{r_j - l_j} P(U = x_1) + ... + x_m P(U = x_m) \right).$$

All jumps are treated in this way, where occasionally some peers of the same upload capacity must be split so there are different numbers of hops to the source.

Before stating the main theorem, we first illustrate, in the following example, how to obtain N_s given C_s and d_p, in a typical streaming session.

Example 1

Suppose the upload capacities of the peers in a streaming session follow a Zipf distribution with average data rate 200 Kbps. Suppose the multimedia source can provide 500 Mbps of serving bandwidth, the download rate for the stream is 250 Kbps, and we expect $N_s = 80,000$ users. Notice that the average upload bandwidth is lower than the average playback data rate, so it may not be possible to serve all the peers in the network.

Using these parameters in the method described above, we hope to organize the peers into an optimal topology and understand the average and maximal delay that would be experienced by these peers. We first truncate and scale a Zipf distribution so that its corresponding mean value equals the average peer upload data rate. In particular, for

$$f(U) = \begin{cases} \frac{k}{u} & \text{if } 0 \leq u \leq m \\ 0 & \text{otherwise} \end{cases}$$

we find k and m such that the integral of $f(U)$ over all values of u is 1 and the mean of U is 200. In this case, $k = 0.1373$ and $m = 1458$ Kbps, so we can write $F(U) = 0.1373 \log(U)$.

Then, by direct calculations, $h_1 = 1 - \frac{500,000/250}{80,000} = 0.975$ and $F^{-1}(0.975) = e^{\frac{0.975}{0.1373}} = 1215$, so we can find

$$A_1 = 80,000 \int_{1215}^{1458} u \frac{0.1373}{u} \mathrm{d}u = 2.67 \times 10^6.$$

Similarly, we find pairs (h_i, A_i) for $i = 2, 3, 4, 5, 6$ to be $(0.8417, 8.29 \times 10^6)$, $(0.4271, 4.81 \times 10^6)$, $(0.1868, 2.03 \times 10^5)$, $(0.1766, 3063)$, $(0.1764, 44)$. We see that even with optimal peer arrangement, since the average upload capacity of the peers is smaller than the playback rate they require, we are unable to serve 18% of the peers.

Note that the above calculation is based on the assumption that only the peers with top 82% of uploading bandwidth are served, so it does not imply that the session can now serve $82\% \times 80,000 = 65,600$ peers. Instead, we must repeat same the procedure with smaller values of N_s. Using the bisection method, we find that the maximum feasible N_s is $10,000$. Alternatively, we could repeat the procedure while adjusting the server bandwidth cost to find the minimum C_s value necessary to support $80,000$ peers.

THEOREM 1. *Suppose the server bandwidth cost is $C_s = g(N_s)$ in an optimal streaming session for N_s served peers each with upload capacity chosen from some distribution $F(U)$. If $\overline{U} > p$, then the number of peers that can be served is unbounded, and*

the corresponding propagation delay to receive the data stream is $\mathrm{O}\left(\log_{\left(\frac{\overline{U}}{p}\right)} \frac{N_s}{g(N_s)}\right)$. *If* $\overline{U} = p$, *then the number of peers that can be served is unbounded, and the corresponding propagation delay to receive the data stream is* $\mathrm{O}\left(\frac{pN_s}{C_s}\right)$. *If* $\overline{U} < p$, *then the number of peers that can be served is bounded, regardless of the allowable delay; however, the maximum number of served peers increases linearly with* C_s.

PROOF. From Lemma 2 and Corollary 1, it is clear that to bound the delay of an optimal network for any upload capacity distribution we should consider the worst-case constant upload capacity at each peer and use all of the source's C_s bandwidth for the peers one hop away. Also recall that when all peers have constant upload capacity, there is no need to calculate the numbers of peers with different uploads, so there would be no iteration to find the maximum feasible value of N_s. Furthermore, we know by Proposition 1 that $d_q = 0$. This means that each peer will be able to serve $\frac{\overline{U}}{p}$ additional peers, and we suppose it serves exactly $\frac{\overline{U}}{p}$ in this optimal session, neglecting the potential suboptimality due to rounding.

We observe fundamentally different performance when $\overline{U} \geq p$ and $\overline{U} < p$. In the first case, the number of peers increases as the number of hops from the server increases. In the second case, the number of peers decreases as the number of hops from the server increases. These cases are considered separately to derive the worst-case delay bound as follows.

Case 1: $\overline{U} > p$

Since adding a peer introduces more bandwidth into the session than it uses, we can express the following relation between N_s and the server bandwidth cost for any number of peers N_s.

$$N_s(d_p) = \sum_{i=0}^{d_p-1} \frac{C_s}{p}\left(\frac{\overline{U}}{p}\right)^i = \frac{C_s}{p}\frac{r^{d_p}-1}{r-1} \text{ for } r = \frac{\overline{U}}{p}, r > 1. \tag{7}$$

Rearranging (7), we see that

$$d_p = \log_{\left(\frac{\overline{U}}{p}\right)}\left\{\frac{N_s p}{C_s}[\frac{\overline{U}}{p} - 1] + 1\right\}. \tag{8}$$

Hence, the server bandwidth cost of $g(N_s)$ corresponds to a delay d_p of $\mathrm{O}\left(\log_{\left(\frac{\overline{U}}{p}\right)} \frac{N_s}{g(N_s)}\right)$ for any general upload capacity distribution.

Case 2: $\overline{U} = p$

In this case, it is easy to see that $d_p = \lceil \frac{pN_s}{C_s} \rceil$, because each peer will entirely serve the next peer in the stream, and hence the optimal session consists of $\lfloor \frac{C_s}{p} \rfloor$ chains radiating from the server. Therefore, the server bandwidth cost of $g(N_s)$ corresponds to a delay d_p of $\mathrm{O}\left(\frac{pN_s}{C_s}\right)$ for any general upload capacity distribution.

Case 3: $\overline{U} < p$

In this case, adding a peer introduces less bandwidth into the session than it uses. If d_p is large, then at some number hops, h_b, from the server, there will not be sufficient bandwidth to support more peers, regardless of delay, i.e., the total upload bandwidth provided by the peers farthest from the source is less than p. In particular, h_b is the smallest value of i such that

$$\frac{C_s}{p}\left(\frac{\overline{U}}{p}\right)^{i-1} < p \Leftrightarrow i > \log_{\left(\frac{\overline{U}}{p}\right)}\left[\frac{p^2}{C_s}\right] + 1,$$

since $\log_a(x) > log_a(y)$ if $a < 1$ and $x < y$. This leads to the number of supported peers

$$\begin{aligned} N_s &= \sum_{i=0}^{\lceil \log_{\left(\frac{\overline{U}}{p}\right)}\left[\frac{p^2}{C_s}\right]+1\rceil} \frac{C_s}{p}\left(\frac{\overline{U}}{p}\right)^i \\ &< \frac{C_s}{p}\frac{\left(\frac{\overline{U}}{p}\right)^{\log_{\left(\frac{\overline{U}}{p}\right)}\left[\frac{p^2}{C_s}\right]+3} - 1}{\left(\frac{\overline{U}}{p}\right)-1} = \frac{\left(\frac{\overline{U}^3}{p^2}\right)-\frac{C_s}{p}}{\left(\frac{\overline{U}}{p}\right)-1}, \end{aligned}$$

and we have $N_s > \frac{\left(\frac{\overline{U}^2}{p}\right)-\frac{C_s}{p}}{\left(\frac{\overline{U}}{p}\right)-1}$, similarly.

If d_p is smaller so that the delay bound is reached before the available bandwidth is entirely used, then the number of peers in the network can be derived as in the previous case:

$$\begin{cases} \frac{\left(\frac{\overline{U}^2}{p}\right)-\frac{C_s}{p}}{\left(\frac{\overline{U}}{p}\right)-1} < N_s < \frac{\left(\frac{\overline{U}^3}{p^2}\right)-\frac{C_s}{p}}{\left(\frac{\overline{U}}{p}\right)-1} \\ \qquad\qquad \text{for } h_b = \log_{\left(\frac{\overline{U}}{p}\right)}\left[\frac{p^2}{C_s}\right] + 1 < d_p, \\ N_s = \frac{C_s}{p}\frac{\left(\frac{\overline{U}}{p}\right)^{d_p}-1}{\left(\frac{\overline{U}}{p}\right)-1} \qquad \text{otherwise.} \end{cases}$$

□

This theorem defines scaling laws that have important practical implications. It allows us to describe the manner in which the delay bound must grow as a function of the number of peers served N_s and the server bandwidth cost C_s. If $\overline{U} \geq p$, an arbitrarily large number of peers can be served because each peer can completely support at least one new peer that enters the system; however, the theorem shows that even in optimal circumstances, the delay necessarily increases as a logarithm for any constant throughput. Alternatively, we see that to achieve similar performance (that is, the same d_p) as N_s increases, the server cost must increase linearly with N_s. In practice, this could be achieved by offering users similar performance at peak user times if they are willing to pay more for the service.[4] If $\overline{U} < p$, then every new peer entering the system needs either support from multiple other peers and/or support from the server. The number of peers that can be supported is necessarily limited because the need for serving bandwidth is higher than the available bandwidth introduced by each peer. Hence, a system designer's only option to support more peers is to burden the source with higher cost.

4. ANALYSIS OF PEER DYNAMICS

In the previous section, we derive the scaling laws of peer-to-peer streaming assuming a static P2P topology. In practice, changes and disruptions are experienced in the P2P network topology due to the arrival and departure of peers, refereed to as *churn*. In this section, we consider churn induced peer dynamics and propose an analytical approach to estimate its effect on the peer service probability, service waiting time, and disconnection period for optimally constructed networks that can be rearranged at will.

To allow tractable analysis, we assume that the arrivals of peers follow a Poisson process and the length of time the peers stay in the session is exponential, which can be extended to an arbitrary lifetime distribution. Furthermore, we suppose the system is ergodic and ignore the upload capacity of particular individuals. Our aim is to calculate the average probability of acceptance of a peer in a large-scale system, assuming that the distribution of upload capacities for the peers in the topology is precisely the same as the probability distribution from which they are chosen[5]. We stress that we are not calculating precise acceptance probabilities of peers into

[4]This type of strategy is used for cellular service plans, for example.

[5]This is true as N becomes large by the Law of Large Numbers.

small networks, since those probabilities depend on the upload capacity of the incoming peer. When we bound P(accepted), this is the average expected acceptance of a peer in a large system.

Recall in Section 3.3 that we define the optimal topology (and hence the number of peers supported for a given delay d_b) for any number of peers N_s that wish to participate in the session. Specifically, we can find the *saturation point* for any upload capacity distribution. At the saturation point, the maximum number of peers can be served at the desired Quality of Service level (within delay bound d_b for server cost C_s) and there are no other peers that wish to join the session. Then, in our discussion of churn, we consider the maximum N_s and represent the topology as an *M/M/*N_s queue of peers that are being served. This is a birth-death queue that is described and solved explicitly in many introductions to queuing theory such as Cooper [12].

4.1 Expected Service Probability

If the new peers wishing to enter the session are blocked, then the queuing system described above is an Erlang Loss System. Let P_j be the probability of j peers being served (receiving media signal) in the system. We have

$$P_j = \frac{\frac{(\lambda/\mu)^j}{j!}}{\sum_{k=0}^{N_s} \frac{(\lambda/\mu)^k}{k!}}, \text{ for } j = 0, 1, 2, ..., N_s \ ,$$

where λ is the mean arrival rate of the peers and $\frac{1}{\mu}$ is the mean service time.

More importantly, we would like to compute the probability that peers attempting to join the session are not served. In this model, we assume that if a peer is refused admission to the session, it leaves and does not retry. Therefore, we express the peer *blocking probability*

$$B(N_s, \lambda/\mu) = \frac{\frac{(\lambda/\mu)^{N_s}}{N_s!}}{\sum_{k=0}^{N_s} \frac{(\lambda/\mu)^k}{k!}}. \tag{9}$$

Note that these results hold for any service-time distribution with finite mean (see Section 3.3 of [12]). In particular, the theory holds true for the Zipf distribution, which has been shown to represent the online lifetimes of human users [13].

Although we have a closed form expression in (9), it is difficult to use this equation because of the need to take factorial for values of k up to N_s. Instead, we bound the probability of a new peer being served as follows. Consider $\frac{1}{B(N_s,\lambda/\mu)}$. Writing out the series backwards, we have

$$\begin{aligned} \frac{1}{B(N_s,\lambda/\mu)} &= 1 + \frac{(\lambda/\mu)^{N_s-1}/(N_s-1)!}{(\lambda/\mu)^{N_s}/N_s!} + \\ &\quad \frac{(\lambda/\mu)^{N_s-2}/(N_s-2)!}{(\lambda/\mu)^{N_s}/N_s!} + ... + \frac{1}{(\lambda/\mu)^{N_s}/N_s!} \\ &= 1 + \frac{N_s}{(\lambda/\mu)} + \frac{N_s(N_s-1)}{(\lambda/\mu)^2} + ... + \frac{N_s!}{(\lambda/\mu)^{N_s}} \end{aligned}$$

When $N_s > \lambda/\mu$, the first $(N_s - \lambda/\mu)$ terms of $\frac{1}{B(N_s,\lambda/\mu)}$ are necessarily ≥ 1. In other words,

$$P(\text{accepted}) = 1 - B(N_s, \lambda/\mu) \geq 1 - \frac{1}{N_s - \lambda/\mu} \ .$$

If, on the other hand, $N_s < \lambda/\mu$, then since there are more peers on average than we expect to be able to serve, the probability of being served is smaller. Clearly $N_s \geq (N_s - i), \forall i \geq 0$. Hence,

$$\frac{1}{B(N_s,\lambda/\mu)} \leq \sum_{k=0}^{N_s} \left(\frac{N_s}{\lambda/\mu}\right)^k = \frac{1 - \left(\frac{N_s}{\lambda/\mu}\right)^{N_s+1}}{1 - \left(\frac{N_s}{\lambda/\mu}\right)} \ ,$$

and therefore

$$P(\text{accepted}) \leq 1 - \frac{1 - \left(\frac{N_s}{\lambda/\mu}\right)}{1 - \left(\frac{N_s}{\lambda/\mu}\right)^{N_s+1}} \ .$$

The next example presents a case study for a typical peer-to-peer live streaming session.

Example 2

Suppose the multimedia streaming session has roughly 3 million participating peers. Each peer tunes in twice per day following a Poisson distribution and continues streaming for an average of 30 minutes each time. Since the sum of Poisson distributions is also Poisson with intensity corresponding to the sum of the component intensities, this corresponds to a session where peers arrive according to a Poisson process with average interarrival time $\frac{1}{\lambda} = \frac{1}{4167}$ minutes. If a peer cannot be served, we assume that an "all servers are busy" message is sent to the user and the request is cleared. Suppose the upload capacity of each peer is constant at rate 300 Kbps. The download rate for the multimedia stream is 250 Kbps, and the users require a delay of no more than 5 seconds $\approx$ 16 hops. Furthermore, the multimedia source can provide 500 Mbps of serving bandwidth.

By Theorem 1, we find that the number of peers N_s that can be served in this session, $N_s(16) = \frac{500,000}{250} \cdot \frac{(300/250)^{16}-1}{(300/250)-1} = 174,884$. We are interested in finding the probability that a user will be served in this session, and we can find this probability directly using Equation (9), or noting that $N_s > \lambda/\mu$, we have a lower bound $P(\text{accepted}) \geq 0.99998$.

4.2 Service Waiting Time

Alternatively, we can consider a system where peers that find all the server busy (cannot be served media within the d_b constraint) join a queue and wait as long as necessary for service.[6] Then we could represent the session as an Erlang Delay System. Again representing the probability of j peers being served as P_j, we have

$$\begin{aligned} P_j &= \frac{(\lambda/\mu)^j}{j!} P_0, \qquad j = 1, 2, ..., N_s - 1 \\ P_j &= \frac{(\lambda/\mu)^j}{N_s! N_s^{j-N_s}} P_0, \qquad j = N_s, N_s + 1, ... \end{aligned}$$

$$\text{for} \qquad P_0 = \left(\sum_{k=0}^{N_s-1} \frac{(\lambda/\mu)^k}{k!} + \sum_{k=N_s}^{\infty} \frac{(\lambda/\mu)^k}{N_s! N_s^{k-N_s}} \right)^{-1} .$$

Clearly, in this case we require that $\lambda/\mu < N_s$; otherwise the queue length becomes infinite and the system becomes unstable.

In this system, we are interested in the waiting times for peers in the queue. Let W be the waiting time for a peer. Then as shown in [12], assuming no peer has priority over another, each peer has probability $1 - P\{W > 0\}$ of being served immediately, where

$$P\{W > 0\} = \frac{(\lambda/\mu)^{N_s}}{N_s!(1 - (\lambda/\mu)/N_s)} P_0. \tag{10}$$

If we suppose that peers are added to the session in the order that they arrived, then it is sufficient to calculate the waiting time given that the peer was not served immediately,

$$P\{W > t | W > 0\} = e^{-[1-(\lambda/\mu)/N_s]N_s\mu t}. \tag{11}$$

Hence, assuming that the arrival rate λ and average service duration $\frac{1}{\mu}$ are given, we can control the queue stability and the waiting time in this system through the parameter N_s. Recall that N_s is the number of peers that can be served in the session and Theorem 1 gives expression for this number in terms of C_s and d_b for the worst-case upload capacities of the peers with mean upload rate $\overline{U}$

[6] We assume that any number of peers can be in the queue awaiting service.

and download rate p. In particular, we can adjust the server bandwidth C_s to allow any arbitrary value of N_s so as to achieve the desired service waiting time distribution.

From (10) and (11) we write an expression for the cumulative distribution of the service waiting time

$$\begin{aligned} F(H \leq t) =& [1 - P(W > 0)] + \\ & P(W > 0)[1 - P(W > t | W > 0)] \\ =& [1 - \frac{(\lambda/\mu)^{N_s}}{N_s!(1 - (\lambda/\mu)/N_s)} P_0] + \\ & (1 - e^{-[1-(\lambda/\mu)/N_s]N_s\mu t})(\frac{(\lambda/\mu)^{N_s}}{N_s!(1 - (\lambda/\mu)/N_s)} P_0) \end{aligned} \quad (12)$$

for $P_0 = \left(\sum_{k=0}^{N_s - 1} \frac{(\lambda/\mu)^k}{k!} + \sum_{k=N_s}^{\infty} \frac{(\lambda/\mu)^k}{N_s! N_s^{k-N_s}} \right)^{-1}$.

With knowledge of the period a peer waits before it can be connected, one could propose that a buffer of data blocks should be stored at each peer to use some of the idle bandwidth available in the session. The collected blocks could be used to begin streaming at the new peer before it is able to join the session, or during subsequent periods of disconnection, the blocks could be played back from the buffer while the peers were searching for new parents. This scenario would minimize playback disruption; however, this type of buffering is limited in the optimal topology to minimize playback delay. Recall that any peer in the middle of the optimal P2P topology is already using all of its upload capacity to serve other peers. The only available spare bandwidth is at leaf peers whose delay is already approaching the maximum delay imposed by the Quality of Service requirement; therefore, if the new peers began streaming, they could not be supported by the leaf peers for extended periods of time.

In a practical P2P topology, however, there might be other idle bandwidth available. Alternatively, in the optimal topology, we may allow leaf peers to serve and exceed the delay constraint. Then, we could use the cumulative distribution function of (12) as the service times and create a new (less severe) Quality of Service level when the topology is approaching its total peer capacity.

5. AFFINITY: PRACTICAL HEURISTICS AND EVALUATIONS

Although this paper is largely theoretical so far, we are fully convinced that the theoretical insights brought forth by this paper have important practical implications. To show how our analytical framework may provide guidelines towards more practical designs, we seek to design and evaluate a simple and realistic heuristic that only depends on local information. Our heuristic is henceforth referred to as *Affinity*, as it is one of the many possibilities of designing practical topology construction protocols that seek to approximate theoretically optimal topologies. The name also captures its essence of categorizing peers with similar upload capacities into roughly the same hierarchy in the topology. The design of Affinity is inspired by our theoretical contributions made so far in this paper. Though discovered independently, we note that Affinity protocol is quite similar to the Dagster [10] protocol in previous works. Dagster's incentive mechanisms to place peers with higher bandwidth closer to the source in the directed acyclic graph indicate the practicality of our suggestions for more efficient topology construction.

5.1 Affinity: a Theory-Inspired Heuristic

Intuitively, since peers with high upload capacities can contribute more to serve other peers in the P2P session, they should be placed closer to the source. In practice, a peer n_a that has just joined the session may communicate with some random sampling of existing peers in the session. If n_a has the highest upload capacity of all the sampled peers, n_a could connect to the source directly to receive the stream. Otherwise, n_a could choose a parent from the sampled peers that has the smallest upload capacity that is still greater than the upload capacity of n_a. This procedure can be repeated at any time when the peer n_a is searching for new parents, for example, if its parent goes offline or has insufficient bandwidth to allow the media to be played at the desired playback rate. We have imposed a general ordering on the collection of peers that is not necessarily optimal, since only a subset of the peers are included in the local tests performed by each peer; however, local tests allow for scalability when the heuristic is implemented.

Although our intuitive placement strategy described above leads to higher numbers of peers that can be served, it also introduces weakness under churn. Since all the upload capacity is used from peers that have more upload capacities to provide, the departure of high-capacity peers can be disruptive to their many children and to the overall performance of the session. One strategy to deal with this problem is to allow each peer to take fewer children and serve at a rate higher than the playback rate until their buffers are full, then to continue sending only at the playback rate. This trades optimality of the number of peers served for tolerance to churn when the peers cannot redistribute themselves arbitrarily.

When a peer joins a P2P streaming session, its place in the topology is found by calculating values of a new parameter δ_k. δ_k is a collective measure of the peers that are k-hops from the multimedia source: it represents the difference between the combined upload capacity of the k-hop peers and the bandwidth needed to serve peers that are $k+1$ hops from the source. Placing the new peer according to the largest $|\delta_k|$ value (as a k-peer if $\delta_k \leq 0$ and as a $(k+1)$-peer if $\delta_k > 0$), the new peer minimizes its burden to the server when it is added to the topology. This strategy is inspired by Corollary 1, from which we learn that the source should serve *only* peers that are one-hop away (directly connected to itself) whenever possible, and by [14].

Further, we wish to place peers with higher upload capacities closer to the source, as directed by Lemma 1. Rather than placing the peer according to the largest $|\delta_k|$ value from the strategy above, we place the peer according to the largest $|\delta_k| \cdot k^{c_1(\overline{U} - u_n)}$ value, where $\overline{U}$ is the mean peer upload capacity, u_n is the upload capacity of the particular peer that has just joined the session, and c_1 is a constant value used to determine the importance of placing higher capacity peers closer to the source.[7] There is no explicit topology maintenance in the Affinity topology. When peers fail in the system, they automatically affect δ_k values, so the topology is naturally maintained as new peers arrive into the system. We also assume that there is no queueing delay, as stated in Proposition 1.

5.2 Affinity vs. Alternatives: Evaluation

We now proceed to evaluate Affinity, in the context of previously proposed strategies of neighbor selection and topology construction. We evaluate two different categories of previous strategies: *randomized* neighbor selection strategies (*e.g.*, CoolStreaming [5]), and *tree-based* neighbor selection strategies (*e.g.*, GridMedia [6]). Both CoolStreaming and GridMedia are excellent examples of operational live media streaming protocols.

Each peer in the *randomized* strategy (called *random*) selects M neighbors uniformly at random from all possible peers in the session. In contrast, the *tree-based* strategy (called *tree*) forms a tree where every peer has $M-1$ children. The initial peer (the source) has M children and all other peers have one parent and $M-1$ children. In *Affinity*, however, a new peer is explicitly placed in the topology at an appropriate location to minimize the

[7] If c_1 is large, then the capacity of the peers will dominate and all of the high capacity peers will be placed closer to the source; however idle bandwidth in the middle of the topology may go unused. If c_1 is smaller, the idle bandwidth in the topology is minimized, but higher capacity peers may be placed lower in the topology.

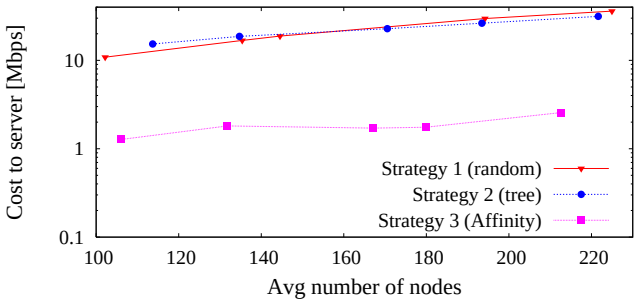

Figure 2: Server cost achieved for different rates of insertion λ, $d_p = 4$ [hops], peers with exponential upload capacities

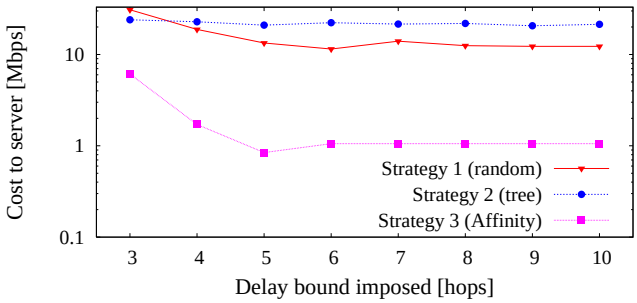

Figure 3: Server cost achieved for different d_p Quality of Service levels, $\lambda = 0.15$ [peers/second], peers with exponential upload capacities

server cost, and also weights the decision so that peers with higher upload capacities are placed higher in the tree.

With a C++-based simulation environment, we have performed comparative evaluations of Affinity, in the context of its two alternatives. We include 130 peers in a P2P streaming session at time zero, each with a Zipf lifetime distribution. The *random* strategy allows peers to choose $M = 4$ partners at random. The delay of a peer is represented by the number of hops on the shortest path from the peer to the source. If the delay of a peer is too long ($> d_p$), that peer must abandon all neighbors and search for new neighbors at the next time-step in the discrete simulation. When peers have no parents to serve them, it is assumed that the source will serve them. Note that it is impossible for peers in the *random* strategy to be more selective about the neighbors they choose, because the distance from the source to a particular peer depends intimately on the neighbors chosen by every other peer. In the simulation, these distances are calculated at the end of a time-step. In reality, it would also take time to perform a check with respect to such distances.

Using the *tree* strategy, each peer has $M - 1 = 3$ children. The topology is reconstructed instantaneously in response to churn, assuming that there is no time for peers to find parents. We wish all peers to be served within the delay bound d_p, so the cost to the server is calculated as $C_s = p\lceil\frac{N_s(M-2)}{(M-1)^{d_p}-1}\rceil$ for a session with N_s peers. *Affinity* inserts new peers into the session according to the weighting $|\delta_k| \cdot k^{(\overline{U}-u_n)}$ described earlier, but will only insert peers at positions $\leq d_p$.

Simulations are run over 500 time-steps (83 minutes, if we assume that a time-step is 10 seconds), with new peers introduced at rate λ and peers expiring according to a Zipf distribution with mean 22 minutes. Upload capacities of peers come from an exponential distribution with mean 300 Kbps, and the playback rate of the media is 225 Kbps. By changing λ, we adjust the average number of peers in the session.

Fig. 2 and Fig. 3 show the bandwidth cost to the server in P2P sessions with different numbers of participating peers and different delay bounds. Clearly, Affinity requires considerably less bandwidth from the server than its alternatives, as *random* and *tree* strategies are not able to take advantage of higher upload capacities at some of the peers. Since the playback rate of the media stream is 225 Kbps and the average upload capacity at a peer is 300 Kbps, it is not surprising that a peer in the *tree* strategy is not able to entirely support its three children and needs assistance from the server to supply the remaining bandwidth. The *random* strategy has the additional problem that for smaller values of the delay bound, a large fraction of the peers cannot be supported by the random neighbors they find.

Since the server costs in the optimal topology are not the same order of magnitude as the practical strategies, we do not include its values in Fig. 2 or Fig. 3. Instead, we present the server costs in the optimal topology in Table 1. Small values are obtained with

Table 1: Server cost of the optimal topology for Figs. 2 and 3

N_s [peers]	C_s [Mbps]	d_p [hops]	C_s [Mbps]
100	1.3×10^{-4}	2	0.03505
130	1.69×10^{-4}	3	0.00269
160	2.08×10^{-4}	4	1.59×10^{-4}
190	2.47×10^{-4}	5	7.52×10^{-6}
220	2.86×10^{-4}	6	2.95×10^{-7}

the optimal topology because the upload capacity distribution has a tail, and all high capacity peers are placed close to the source, so for the first few hops it appears as if the mean upload capacity is greater than the playback rate. After that point, the number of peers served decreases at each hop, but most of the peers have already been served. Furthermore, we do not account for rounding to an integer number of peers in the optimal topology. As a result, the C_s of the optimal scheme with the network conditions of Fig. 2 is less than 1 Kbps. If the server was allocated as much bandwidth as Affinity for 220 peers (roughly 1.2 Mbps), then the optimal topology would support approximately $1\frac{1}{2}$ million peers, provided that they could be ordered from the source according to the size of their upload capacities regardless of churn.

Fig. 3 uses a constant peer arrival rate so that the number of peers in the network has only small variations, but allows for longer maximum delay as the media stream is delivered to a peer. As we might expect, by relaxing the delay constraint, the server cost can be reduced. The reduction in server cost as delay bounds are relaxed is most prominent with the optimal scheme (by several orders of magnitude), as all of the upload capacities of all the intermediate peers are fully utilized. The practical strategies are not able to decrease as quickly as the optimal one, since it is not feasible for them to arrange all of the peers in precise decreasing order, and they always have an integer number of peers at each hop from the source. Still, when compared with *random* and *tree* strategies, Affinity performs significantly better.

5.3 Peer Dynamics with Bounded Server Cost

We now evaluate how new peers are admitted into the P2P streaming session when a limited server bandwidth is imposed. In this experiment, we assume that the server cannot provide more than 4 Mbps. If more bandwidth is requested from a peer, it is not admitted to the session. Furthermore, all peers have uniform upload capacities $u = 200$ Kbps, and the playback rate for the media is $p = 225$ Kbps. The arrival of new peers follows a Poisson process with a combined rate of 22.4 [peers/second], and the service times follow the Zipf distribution with a mean service time of $\frac{1}{\mu} = 5$ [seconds/peer]. We vary the maximum permitted delay d_b, which in turn changes N_s, but $N_s < \lambda/\mu = 112$ in all cases. As discussed in Section 4, if $N_s > \lambda/\mu$, the probability of a peer being admitted into the session in the optimal topology is close to 1. Hence, it is more interesting to consider the case where

Table 2: Number of peers served in the P2P session

d_b [hops]	actual N_s [peers]	optimal N_s [peers]
2	33	34
3	47	48
4	59	60
5	69	71
6	78	81
7	86	90
8	93	98
9	99	104

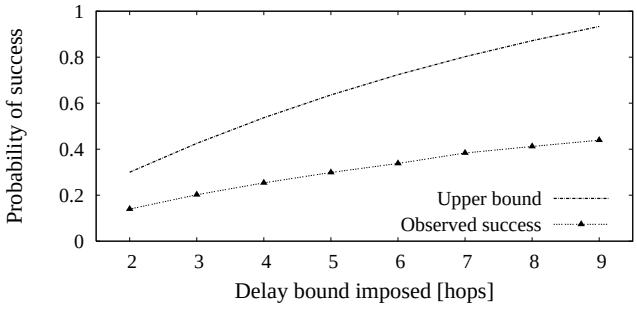

Figure 4: Probability that peers will be admitted into the P2P streaming session given a limited server bandwidth $C_s = 4$ Mbps.

$N_s < \lambda/\mu$, in which peers are sometimes not successful to join the session.

We tabulate in Table 2 the optimal N_s calculated using (7). It is compared with the actual number of peers that are served on average in simulation. Since all peers have uniform upload capacities, the simulated session performs near-optimally. We observe a small discrepancy between the optimal and simulated values due to the requirement of an integer number of peers at any given number of hops from the source in a practical session. Furthermore, Section 4.1 has described a conservative upper bound for the probability of admission of peers in an optimal topology. Clearly it is also an upper bound for the admission of peers in a practical session. In Fig. 4, we compare the derived upper bound for the probability of acceptance of a peer into the session with the actual probability of new peer admission. This figure shows that the derived upper bound follows the actual trend observed. In particular, this curve would give a network designer insight into the maximum delay that should be tolerated in order to achieve a desired success of peer acceptance into a streaming session. As we have seen many times before, by relaxing the delay constraint, better network performance can be observed; however, the improvements are limited as the delay constraint grows to higher values.

6. CONCLUDING REMARKS

In this paper, we seek to study the scaling laws of constructing high-quality peer-to-peer DAG topologies for live multimedia streaming. We have proposed a theoretical framework to construct optimal peer-to-peer topologies that minimize bandwidth costs on streaming servers, and to investigate the tradeoffs among server costs, scalability and delay bounds in peer-to-peer streaming sessions. We have considered arbitrary distributions of peer upload capacities, and derived an explicit closed-form solution for the resource-performance tradeoff in the worst case. As peer-to-peer topologies are inherently dynamic as peers join and leave, we have also proposed an analytical framework to estimate the effect of peer churn on the peer service probability and service waiting time. We are fully convinced that our theoretical insights introduced in this paper have important practical implications. We introduce *Affinity* as a proof-of-concept showcase of heuristics that can be practically implemented yet influenced by our theoretical conclusions. As simple as it is, Affinity is a better performer than alternative tree-based or randomized topology construction strategies in previous work in our evaluations. In our future work, we are interested in studying the complete spectrum of realistic, efficient and simple heuristics that further utilize our theoretical conclusions, and put them to good use in real-world live peer-to-peer streaming implementations.

7. REFERENCES

[1] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O. Jr., "Overcast: Reliable Multicasting with an Overlay Network," in *Proc. Operating Systems Design and Implementation*, pp. 292–301, October 2000.

[2] 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.

[3] D. Kostić, 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.

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

[5] 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*, March 2005.

[6] 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*, November 2005.

[7] J. C. V. Venkataraman, P. Francis, "Chunkyspread: Multi-tree Unstructured Peer-to-Peer Multicast," in *Proc. 5th International Workshop on Peer-to-Peer Systems*, February 2006.

[8] V. Vishnumurthy and P. Francis, "On Heterogeneous Overlay Construction and Random Node Selection in Unstructured P2P Networks," in *Proc. INFOCOM*, April 2006.

[9] A. Rodriguez, D. Kostić, and A. Vahdat, "Scalability in Adaptive Multi-Metric Overlays," in *Proc. 24th International Conference on Distributed Computing Systems*, pp. 112–121, March 2004.

[10] W. Ooi, "Dagster: Contributor-aware end-host multicast for media streaming in heterogeneous environment," in *Proc. SPIE Multimedia Computing and Networking (MMCN)*, January 2005.

[11] J. Liang and K. Nahrstedt, "DagStream: Locality Aware and Failure Resilient Peer-to-Peer Streaming," in *SPIE Multimedia Computing and Networking (MMCN)*, January 2006.

[12] R. B. Cooper, *Introduction to Queueing Theory*. Elsevier North Holland, Inc., 1981.

[13] 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*, pp. 107–120, September 2004.

[14] T. Small, B. Li, and B. Liang, "Outreach: Peer-to-Peer Topology Construction towards Minimized Server Bandwidth Costs," in *IEEE Journal on Selected Areas in Communications, Special Issue on Peer-to-Peer Communications and Applications, First Quarter, 2007*.