Performance Bounds for Peer-Assisted Live Streaming

Shao Liu*, Rui Zhang-Shen*, Wenjie Jiang†, Jennifer Rexford†, Mung Chiang*
*Department of Electrical Engineering, and †Department of Computer Science
Princeton University
{shaoliu, rz, wenjiej, jrex, chiangm}@princeton.edu

ABSTRACT

Peer-assisted streaming is a promising way for service providers to offer high-quality IPTV to consumers at reasonable cost. In peer-assisted streaming, the peers exchange video chunks with one another, and receive additional data from the central server as needed. In this paper, we analyze how to provision resources for the streaming system, in terms of the server capacity, the video quality, and the depth of the distribution trees that deliver the content. We derive the performance bounds for minimum server load, maximum streaming rate, and minimum tree depth under different peer selection constraints. Furthermore, we show that our performance bounds are actually tight, by presenting algorithms for constructing trees that achieve our bounds.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Performance attributes]

General Terms

Algorithms, Design, Performance

Keywords

peer-to-peer, video, streaming, tree construction, IPTV

1. INTRODUCTION

The rapid growth in residential broadband capacity is enabling the delivery of high-quality video over the Internet. Server-based video delivery can provide performance guarantees, but the server infrastructure is expensive and may not scale well. Peer-to-peer (P2P) technology, already widely used for file-sharing applications, has the potential to reduce server and network load by allowing consumers to download live video content from each other [1, 2, 3, 4]. However, existing P2P streaming applications suffer from low-quality video, periodic hiccups, and high delay [2, 3], making it difficult for service providers to leverage the technology directly in commercial offerings.

In this paper, we argue that service providers can deploy a *peer-assisted* streaming architecture to offer high-quality video at a reasonable cost. Fortunately, in carrier-based streaming services, the provider has greater control over the peers, which may run on dedicated equipment like set-top boxes [5]. This substantially reduces the churn introduced by peers joining and leaving the system; in fact, the provider may have the peers continue to distribute a stream after users stop watching the video. In addition, the provider can influence, or even control, how the peers are organized into distribution trees for delivering the content. When the peers do not have enough bandwidth to distribute all of the data, the provider can have the server transmit more than one copy of the stream, as needed. This allows the provider to ensure that all peers can download the content at the target streaming rate. We believe that such a hybrid, peer-assisted streaming architecture combines the best features of server-based and peer-based solutions.

In an emerging field like peer-assisted streaming, we need to lay the theoretical foundations that can drive the design of scalable systems in the future. In this paper, we derive performance bounds and present optimal tree-construction algorithms that service providers can use to provision scalable, peer-assisted streaming services. Our analytical models focus on three main metrics:

- **Server capacity:** To reduce cost, the service provider wants to minimize the upload rate of the server, while still ensuring that all peers can receive the live stream at the target rate.
- **Streaming rate:** To offer high video quality, the service provider wants the system to deliver video content at a high rate, subject to the capacity of the server and the peers.
- **Tree depth:** To improve robustness and minimize latency, the service provider needs distribution trees that limit the number of intermediate peers between server and consumer.

We optimize these three metrics as a function of the number of peers and their upload capacities, subject to different constraints on how peers connect to each other. In particular, we study the effects of *restrictions on the number of downstream neighbors a peer can serve*, or *the outgoing degree of a peer*, to limit the state that each peer must store and maintain, which is especially important as the system grows large. Our contributions in this paper are two-fold:

Exploring a rich design space: We model practical constraints on the outgoing degrees of peers. We explore three types of constraints on this problem, as illustrated by the rows in Table 1. We first explore the two extreme design points where a peer can communicate with any neighbor, or only one neighbor. Then, we model arbitrary constraints on the outgoing degrees.

Tight bounds on performance metrics: We derive the optimal values for these metrics, and provide tree-construction algorithms

	Min. server load (s_{min})	Max. streaming rate (r_{max})	Min. tree depth (D_{min})
Unconstrained peer selection (Sec. 3)	$\surd\surd$	$\surd\surd$ [6, 7]	$\surd\surd$
Single downstream peer (Sec. 4)	$\surd\surd$	$\surd\surd$	$\surd\surd$
Multiple downstream peers (Sec. 5)	$\surd\surd$	$\surd\surd$	$\surd$

Table 1: Summary of results ("$\surd$" indicates upper and lower bounds, and "$\surd\surd$" indicates an exact optimal value).

r:	Data rate of the streaming video.
S:	Server capacity.
s:	Actual server load.
N:	Total number of users in the streaming system.
$\mathcal{N}$:	Index set of all the users,i.e., $\{1,2,\cdots,N\}$
U_i:	Uplink capacity of user i.
u_i :	User i's aggregate uploading rate.
d_i :	User i's aggregate downloading rate.
x_{ij} :	Streaming rate from user i to user j.
s_i:	Streaming rate from the server to user i.
L:	Total number of trees in a multi-tree.
$y^{(l)}$:	Rate of substream delivered by tree l.
$E^{(l)}$:	Set of leaf users in tree l.
$F^{(l)}$:	Set of non-leaf users in tree l.
$m_i^{(l)}$:	Fanout or outgoing degree of user i in tree l.
$D^{(l)}$:	Depth of tree l.
D:	$D = \max_l D^{(l)}$ is the depth of the multi-tree.
M:	Maximum fanout of a tree.
M_t:	Maximum fanout of a multi-tree.

Table 2: Main notation used in this paper.

that achieve the optimal values, for minimum server load, maximum streaming rate, and minimum tree depth, as illustrated by the columns in Table 1. Our bounds are tight, except for the challenging case of minimizing tree depth under arbitrary constraints on the outgoing degree of each peer.

The remainder of the paper is organized as follows. Section 2 presents our model and notation. Sections 3, 4, and 5 present the bounds and algorithms for the three rows of Table 1, respectively. Section 6 presents related work, and Section 7 concludes with a discussion of future research directions. Without explicit explanation, the proofs of all theorems are given in the main text, while the proofs of all lemmas are postponed to the Appendix.

2. PEER-ASSISTED STREAMING MODEL

In this section, we present our model of peer-assisted streaming, including the underlying assumptions and the notation (summarized in Table 2).

2.1 Assumptions

In this paper, we analyze a peer-assisted live streaming application managed by a service provider. Therefore we make the following assumptions in our analysis:

- Peer churn can be ignored because provider-controlled set-top boxes can be always on in the timescale of our problem.
- Each user receives each bit of the video stream only once. Any repetitive downloading would make the system inefficient.
- Uplink bandwidth is the only bottleneck, because the backbone network is well-provisioned and the residential users have asymmetric access bandwidths. In addition, if the downlinks were the bottleneck, the users would not be able to receive the entire video.
- We assume that the server can upload to as many users as needed, with no constraint on the number of simultaneous connections. In practice, the server may be implemented on a collection of machines that distribute the video content to different subsets of the users.
- We do not consider the location of the peers or the topology of the network, though a real system should take these issues into account to reduce congestion and delay.

When these assumptions hold, most bounds we derive in this paper are tight. In a more general setting, our bounds still hold but may not be tight; that is, our models still provide a lower bound for server load and tree depth, and an upper bound on the achievable streaming rate.

2.2 Constraints on Distributing the Stream

We consider a server that generates a video stream of rate r and N users who want to watch it. Throughout the paper we use user, peer, and node interchangeably. We treat the data stream as fluid so it is continuous and can be infinitely divided into *substreams* to any precision without loss of video quality or any overhead.

Let S denote the server capacity and s the actual server load. Denote the uplink capacity of user i by U_i, the upload rate of user i by u_i, and the download rate of user i by d_i, for $i = 1,2,\cdots,N$. When convenient, we use $\mathcal{N}$ to represent the set of users. For simplicity of presentation, we assume that the uses are indexed according to their uplink capacities, i.e., $U_1 \geq U_2 \geq \cdots \geq U_N$.

Let x_{ij} be the rate at which user i uploads to user j. For convenience define $x_{ii} = 0$ for all i. Let s_i be the rate at which user i downloads from the server. We then have the following:

$$s = \sum_{i=1}^{N} s_i \leq S \quad (1)$$

$$u_i = \sum_{j=1}^{N} x_{ij} \leq U_i, \text{for } i = 1,\cdots,N \quad (2)$$

$$d_i = s_i + \sum_{j=1}^{N} x_{ji}, \text{for } i = 1,\cdots,N \quad (3)$$

Since users never download any data more than once, we have

$$d_i = r. \quad (4)$$

User i should not send back anything it receives from user j, so we also have

$$x_{ij} \leq d_i - x_{ji} \quad (5)$$

One key feature of a P2P system is the conservation of flows, i.e., the total upload rate equals to the total download rate. Combining

Equations (1)-(4), we have

$$s + \sum_{i=1}^{N} u_i = \sum_{i=1}^{N} s_i + \sum_{i=1}^{N} \sum_{j=1}^{N} x_{ij} = \sum_{i=1}^{N} d_i = Nr \tag{6}$$

The above equations and inequalities are *necessary* conditions for the system to support a stream of rate r, and they will be used throughout the paper.

2.3 Distribution Trees for Stream Delivery

For *sufficient* conditions we can specify how each substream is distributed among all the users. If we trace the forwarding of an infinitesimal substream, the delivery paths should form a tree rooted at the server and consisting of all the users, with each user appearing in the tree exactly once. We call this a *substream tree*, or simply a *tree*. Other substreams may traverse different trees. In each tree, a peer receives the substream from a single *parent* and uploads the substream to zero or more *children*.

Suppose there are L substream trees, and tree l is responsible for delivering a substream of rate $y^{(l)}$. This means each edge in tree l represents a flow of rate $y^{(l)}$. The superposition of these substream trees form a *multi-tree*. In order for the multi-tree to support a video stream of rate r, we need

$$\sum_{l=1}^{L} y^{(l)} = r. \tag{7}$$

Let $m_i^{(l)}$ and $m_s^{(l)}$ be the *fanouts*, or the *outgoing degrees*, which are the numbers of children, of user i and the server, respectively, in tree l. Denote by $F^{(l)}$ the set of non-leaf users in tree l, and $E^{(l)}$ the set of leaf users in tree l. Then we have

$$\begin{array}{rcl} u_i & = & \sum_{l:i\in F^{(l)}} m_i^{(l)} y^{(l)}, \ \forall i = 1, \cdots, N \\ s & = & \sum_{l=1}^{L} m_s^{(l)} y^{(l)}. \end{array} \tag{8}$$

Let $D^{(l)}$ be the *depth* of tree l, or the maximum number of hops in tree l from the server to the users, and let $D = \max_l D^{(l)}$ be the maximum depth of all the trees, or the *multi-tree depth*. Then D is the maximum number of hops a substream has to traverse to reach all users. A higher D means a larger number of hops, which potentially increases the end-to-end delay from the original source (the server) to a receiver. Furthermore, a particular peer stops receiving a substream when any of the ancestor nodes fail; thus a larger D (i.e., more ancestors per peer) also negatively affect the robustness of the content delivery tree structure. So we would like D as small as possible.

Let M be the *substream tree degree bound*, or the maximum allowed number of children a user has in any substream tree, and let M_t be the *multi-tree degree bound*, or the maximum allowed number of distinct children a user has in the multi-tree. M reflects the complexity in constructing the substream trees; while M_t reflects the total number of downstream peers a user has to serve. So clearly we would like M and M_t small. But doing so may increase the tree depth D, so there is a tradeoff between system complexity (M and M_t) and performance (D). Note that for the same value, M_t gives a stronger constraint, because $M_t \le a$ implies $M \le a$.

3. UNCONSTRAINED PEER SELECTION

In this section, we study the simplest case where there is no limit on the number of children each peer can have. This means, a user can forward a substream it receives to any number of peers. The maximum supported rate in this case has been studied in [6, 7]:

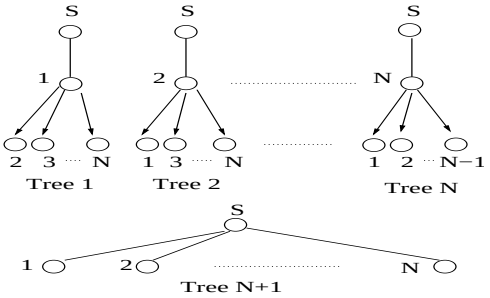

Figure 1: The content distribution multi-tree to minimize the server load in Theorem 1. Case 1 uses tree 1 to N, and case 2 uses tree 1 to $N+1$. For both cases, the multi-tree depth is 2.

$$r_{max}(S) = \min\left(S, \frac{1}{N}\left(S + \sum_{i=1}^{N} U_i\right)\right). \tag{9}$$

We now answer the other two questions in the first row of Table 1. The following theorem states the minimum server load as well as the minimum multi-tree depth needed, when supporting a streaming rate r:

Theorem 1. *Let $s_{min}(r)$ denote the minimum server load that can support a stream of rate r, then*

$$s_{min}(r) = \max\left(r, Nr - \sum_{i=1}^{N} U_i\right), \tag{10}$$

and the minimum server load can be achieved by a multi-tree with depth $D = 2$.

Proof. Our proof technique is inspired by [6]. We first show that Equation (10) is a lower bound, and then show it is achievable by trees of depth 2.

The server has to supply the original stream, so $s \ge r$. From (6) we have

$$s = Nr - \sum_{i=1}^{N} u_i \ge Nr - \sum_{i=1}^{N} U_i.$$

Therefore

$$s \ge \max\left(r, Nr - \sum_{i=1}^{N} U_i\right).$$

To show that the bound is achievable we consider two cases, and construct a multi-tree of depth 2 for each case. Case 1 is when the rate r is small enough to be supported by the peers' upload capacity and the server does not need to inject extra bandwidth into the system, i.e., $s = r$; Case 2 is when the rate is high enough that the server must inject extra bandwidth, i.e., $s = Nr - \sum_{i=1}^{N} U_i > r$.

Case 1: $r \ge Nr - \sum_{i=1}^{N} U_i$, or

$$r \le \frac{1}{N-1} \sum_{i=1}^{N} U_i. \tag{11}$$

We divide the stream of rate r into N substreams, with the i-th substream having rate $\frac{U_i}{\sum_{j=1}^{N} U_j} r$. The server sends substream i to user i, who distributes it to all the other $N-1$ users. The multi-tree implementation is shown in Figure 1 and it uses trees 1 through N.

Now we check that the uplink capacity constraints are satisfied. The server uploads each substream only once, so its upload rate is $s = r$. The upload rate of user i is

$$u_i = \frac{U_i}{\sum_{j=1}^{N} U_j} r \cdot (N-1) \le \frac{U_i}{\sum_{j=1}^{N} U_j} \cdot \sum_{j=1}^{N} U_j = U_i ,$$

where the inequality is due to (11).

Case 2: $r < Nr - \sum_{i=1}^{N} U_i$, or

$$r > \frac{1}{N-1} \sum_{i=1}^{N} U_i .$$

We divide the stream of rate r into $N+1$ substreams, with the i-th substream having rate $U_i/(N-1)$ for $i = 1, 2, \cdots, N$ and the $(N+1)$-th stream having rate $r - \sum_{i=1}^{N} U_i/(N-1)$. The server sends substream i to user i, for $i = 1, 2, \cdots, N$, who distributes it to all other $N-1$ users. In addition, the server sends substream $(N+1)$ to all users. The multi-tree implementation is demonstrated in Figure 1 and it uses trees 1 through $(N+1)$.

Now we check that the uplink capacity constraints are satisfied. For user i, the upload bandwidth is

$$u_i = U_i/(N-1) \cdot (N-1) = U_i .$$

The server sends each of the first N substreams once and sends the last substream N times. So

$$s = \sum_{i=1}^{N} \frac{U_i}{N-1} + N \left(r - \sum_{i=1}^{N} \frac{U_i}{N-1} \right) = Nr - \sum_{i=1}^{N} U_i .$$

In both cases each substream traverses at most two hops so $D = 2$. □

Naturally, a system with unconstrained peer selection supports the highest streaming rate, and given a rate, it requires the lowest server load and the lowest multi-tree depth. Therefore, Theorem 1 provides a benchmark for the cases in the rest of this paper. Note that $D = 2$ is the minimum multi-tree depth that can utilize peer uplink bandwidth, since if $D = 1$, peers only download from the server and do not upload to each other.

4. SINGLE DOWNSTREAM PEER

Section 3 studied unconstrained peer selection, i.e., $M = N-1$. In a real system, we want to limit the number of peers because N can be large. A node should be allowed to upload to only a small number of peers to limit the amount of states it has to store and maintain.[1] So this section studies the other extreme case of single peer selection, i.e., $M = 1$ or $M_t = 1$.

The constraint $M = 1$ means that a user can upload to one peer in each substream tree, but in the multi-tree (superposition of the substream trees) can upload to many peers. The constraint $M_t = 1$ means that a user can upload to only one peer in the multi-tree. So, $M_t = 1$ is a stronger constraint. Luckily, for minimum server load (Section 4.1) and maximum streaming rate (Section 4.2), the two constraints have the same optimal value. But for minimum tree depth, they give different answers (Section 4.3 for $M = 1$ and Section 4.4 for $M_t = 1$).

4.1 Minimum Server Load

We first determine the minimum server load required to support a streaming rate r. Similar to the analysis in the unconstrained case, in order to achieve the minimum server load, we need to maximize each user's uploading rate. Since each user can upload to only one peer, the maximum upload rate of user i is

$$\hat{U}_i := \min(r, U_i) . \tag{12}$$

Therefore, similar to the proof in Theorem 1, the lower bound for s_{min} under the $M = 1$ constraint is:

$$s_{min}(r) \ge \max \left(r, Nr - \sum_{i=1}^{N} \hat{U}_i \right) = \max \left(r, \sum_{i=1}^{N} (r - U_i)^+ \right) , \tag{13}$$

where $a^+ := \max(0, a)$.

This bound is in fact achievable so we have the theorem:

Theorem 2. *When a user can upload to at most one peer in each substream tree, to support a streaming rate r, the minimum server load is*

$$s_{min}(r) = \max \left(r, \sum_{i=1}^{N} (r - U_i)^+ \right) . \tag{14}$$

In particular, if $r \ge U_1$, the $M = 1$ constraint does not affect the minimum server load.

Proof. We show that the bound is achievable by considering two cases.

Case 1: $r \le U_{N-1}$. We have $s_{min}(r) = r$, which is achievable by a single tree in a chain topology *server* $\to 1 \to 2 \to \cdots \to N-1 \to N$ with rate r.

Case 2: $r > U_{N-1}$. Suppose r lies between U_{k-1} and U_k, i.e., $U_{k-1} > r \ge U_k$ (if $k = 1$ then we have $r \ge U_1$). Then, $\hat{U}_i = r$, $\forall i < k$, and $\hat{U}_i = U_i$, $\forall i \ge k$. This means that the $M = 1$ constraint reduces the maximum upload rate for peers 1 to $k-1$ (if $k > 1$), but not for peers k to N, and the lower bound for s_{min} is $\max\left(r, \sum_{i=k}^{N} (r - U_i)\right)$.

We show that this bound is achievable by a multi-tree algorithm with the ring topology shown in Figure 2. In this ring, we call node $i-1$ the *predecessor* of node i, and $i+1$ the *successor* of i.[2] Each node uploads only to its successor, and downloads from its predecessor, and if necessary, the server as well. Node i's download rate is $d_i = r$, $\forall i = 1, 2, \cdots, N$. For the download/upload rates, we

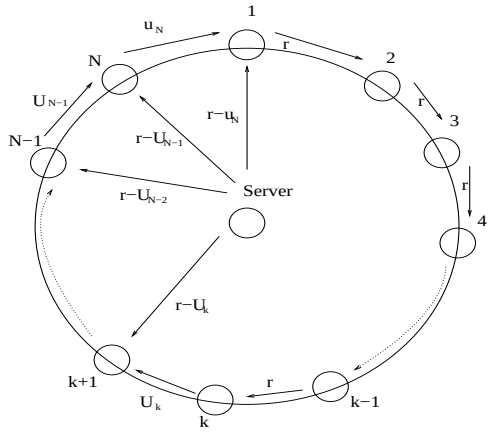

Figure 2: The optimal logical topology for single downloading peer constraint.

[1] For example, in BitTorrent [8], a node uploads to at most five peers simultaneously.

[2] For the compactness of presentation, let $i-1$ represent N, when $i = 1$; and let $i+1$ represent 1 when $i = N$.

The Bottleneck Removal Algorithm

(1) $u_i^{(0)} = u_i = \min(r, U_i)$, $s_i^{(0)} = r - u_{i-1}, \quad \forall i$
(2) for $l = 0$ to $L-1$ do
(3) $\quad k^{(l)} = \arg\min\{u_i^{(l)}, s_i^{(l)} : u_i^{(l)} > 0, s_i^{(l)} > 0\}$
(4) $\quad$ Case (I): $\min\{u_i^{(l)}, s_i^{(l)} : u_i^{(l)} > 0, s_i^{(l)} > 0\} = u_k^{(l)}$
(5) $\quad\quad$ Let $y^{(l)} = u_k^{(l)}$.
(6) $\quad\quad$ Let $p^{(l)}$ be $k^{(l)}$'s closest ancestor, or $p^{(l)} = k^{(l)}$, s.t. $s_{p^{(l)}}^{(l)} > 0$. Construct a tree branch as $s \to p^{(l)} \to \cdots \to k^{(l)} \to \cdots \to q^{(l)}$, where $q^{(l)} = p^{(l)} - 1$ or $u_{q^{(l)}}^{(l)} = 0$.
(7) $\quad\quad$ If $q^{(l)} \neq p^{(l)} - 1$, let $k^{(l)} = q^{(l)} + 1$, repeat (6).
(8) $\quad\quad$ Assign the constructed tree l a streaming rate of $y^{(l)}$.
(9) $\quad$ Case (II): $\min\{u_i^{(l)}, s_i^{(l)} : u_i^{(l)} > 0, s_i^{(l)} > 0\} = s_k^{(l)}$
(10) $\quad\quad$ Let $y^{(l)} = s_k^{(l)}$.
(11) $\quad\quad$ Let $p^{(l)}$ be $k^{(l)}$'s furthest descendant, or $p^{(l)} = k^{(l)}$, s.t. $u_{p^{(l)}}^{(l)} = 0$. Construct a tree branch as $s \to k^{(l)} \to k^{(l)} + 1 \to \cdots \to p^{(l)}$.
(12) $\quad\quad$ If $p^{(l)} \neq k^{(l)} - 1$, let $k^{(l)} = p^{(l)} + 1$, repeat (11).
(13) $\quad\quad$ Assign the constructed tree l a streaming rate of $y^{(l)}$.
(14) $\quad$ for $i = 1$ to N do
(15) $\quad\quad$ $u_i^{(l+1)} = u_i^{(l)} - y^{(l)}$, if $(i, i+1)$ is in tree l.
$\quad\quad$ $s_i^{(l+1)} = s_i^{(l)} - y^{(l)}$, if (s, i) is in tree l.
(16) $\quad$ end for
(15) end for

Table 3: The Bottleneck Removal algorithm for the construction of a multi-tree to achieve the lower bound on the minimum server load.

have

$$\begin{aligned} u_{i-1} &= x_{i-1,i} = r \le U_{i-1} \text{ and } s_i = 0, \forall i = 2, \cdots k \text{ (if } k > 1), \\ u_{i-1} &= x_{i-1,i} = U_{i-1} \text{ and } s_i = r - U_{i-1}, \forall i = k+1, \cdots, N, \\ u_N &= x_{N,1} = \begin{cases} U_N & \text{if } \sum_{i=k}^{N}(r - U_i) > r, \\ \sum_{i=k}^{N-1}(r - U_i) & \text{if } \sum_{i=k}^{N}(r - U_i) \le r, \end{cases} \\ s_1 &= r - u_N. \end{aligned}$$

There are two cases for u_N because if $\sum_{i=k}^{N}(r - U_i) > r$, then $s > r$ and all the peers from k to N should upload at their capacities in order to minimize the server load, and thus $u_N = U_N$, while if $\sum_{i=k}^{N}(r - U_i) \le r$, then the server load is bounded by the streaming rate r, and thus peer N need not upload at capacity.

Now all peers have the download rate r, and the server load is

$$s = \sum_{i=1}^{N} s_i = \max\left(r, \sum_{i=k}^{N}(r - U_i)\right),$$

which is the lower bound on s_{min}. So we only need to design a multi-tree that achieves this rate allocation. Figure 3 illustrates an example of constructing a multi-tree for a streaming system with $N = 6$ users, by applying the Bottleneck Removal algorithm described in detail below.

The Bottleneck Removal Multi-tree Construction Algorithm:

Consider the multi-tree construction algorithm in Table 3. In each iteration we construct one tree that consumes the *bottleneck* link capacity on the ring or the spoke. The residual capacity of that edge becomes zero and will be omitted in the following iterations. The algorithm terminates after at most N iterations and produces at

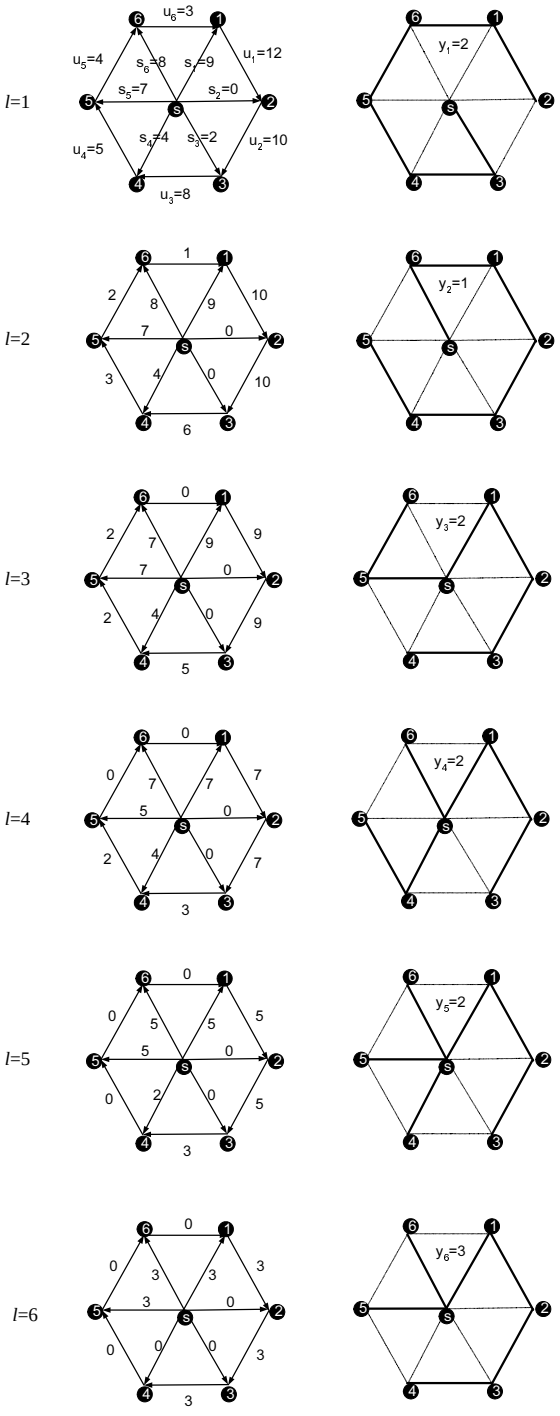

Figure 3: An example to illustrate the construction of a multi-tree using the Bottleneck Removal algorithm. When $l = 1$, the bottleneck is $(s, 3)$ and case II applies ($k = 3, p = 2$). We build a tree branch as $s \to 3 \to \ldots \to 2$, and the residual capacities of edges on the branch are reduced by $y_1 = 2$. When $l = 3$, the bottleneck is $(5, 6)$ and case I applies ($k = 5, p = 5, q = 6$). We build a tree branch from s to 5 and get stuck at 6. We forward to the next user 1, and repeat the same process, which results in another tree branch $s \to 1 \to 2 \to 3 \to 4$. Note that in each iteration, all users will be covered in the spanning tree rooted at s.

most N substream trees. The residual capacities on the ring and the spoke are initialized to the rates in Figure 2. In each iteration we pick an edge with the minimum residual capacity. Let $u_i^{(l)}$ denote the residual bandwidth on edge $(i, i+1)$ after the l-th iteration and $s_i^{(l)}$ denote the residual bandwidth of edge $(server, i)$ after the l-th iteration. For convenience, we define $u_i^{(0)} = u_i$ and $s_i^{(0)} = s_i$.

Consider the following two cases: (I) If the bottleneck edge is on the ring, we search counter-clockwise for the first user (say $p^{(l)}$) to whom the server's residual bandwidth is non-zero. We build a tree branch starting from s to $p^{(l)}$ and stop at a user (say $q^{(l)}$) where we cannot expand anymore. We repeat the same process from the next user $q^{(l)}+1$, till all the users are spanned. (II) If the bottleneck edge is on the spoke, we build a branch of the tree starting from the spoke until we cannot expand anymore. We move on from where we get stuck to the next user and repeat the same process. After building a tree, the residual capacities on the ring and the radius are deducted by the streaming rate of the tree.

In the ring topology (Figure 2), and the multi-tree we constructed above, each user has at most one child, therefore satisfying the $M = 1$ constraint. The correctness of our multi-tree construction is shown by the following lemma:

Lemma 1. *The rate allocation in the ring topology, as illustrated in Figure 2, can be achieved by applying the Bottleneck Removal algorithm to construct a multi-tree within N iterations.*

From the proof in the Appendix, the Bottleneck Removal algorithm runs in polynomial time. This lemma shows that the lower bound of s_{min} in (13) is achievable, proving Theorem 2. □

Remark: Theorem 2 gives the minimum server load under the $M = 1$ constraint. In either the chain or the ring topology which achieves the minimum server load, the multi-tree maximum fanout is also 1. Since under the stronger $M_t = 1$ constraint, the minimum server load should be greater than or equal to that under the $M = 1$ constraint, this implies that when $M_t = 1$, the minimum server load is also given by Equation (14).

4.2 Maximum Streaming Rate

Finding the maximum supported rate is in some sense the dual of finding the minimum server load. Similar to minimizing the server load, to achieve the maximum streaming rate, the peer uplink bandwidth should be utilized as much as possible. Under the $M = 1$ constraint, the upper bound for a supported streaming rate r is

$$r \le \min(S, \frac{1}{N}(S + \sum_{i=1}^{N} \hat{U}_i)), \tag{15}$$

where the maximum upload rate of user i is $\hat{U}_i$ given in (12).

The bound is achievable but $\hat{U}_i$ is a function of r, so the expression for r is not straightforward. We have the following theorem:

Theorem 3. *When a user can upload to at most one peer in each substream tree, for a given server upload capacity S, the maximum supported streaming rate is*

$$r_{max}(S) = \begin{cases} S & \text{if } S \le U_N, \\ \min(S, g(k^*)) & \text{if } S > U_N, \end{cases} \tag{16}$$

where

$$g(k) := \frac{S + \sum_{i=k}^{N} U_i}{N - k + 1}. \tag{17}$$

and k^* is the minimum k such that

$$S \ge \sum_{i=k}^{N} (U_k - U_i) \text{ and } S \ge U_k. \tag{18}$$

In particular, if $k^* = 1$, then the $M = 1$ constraint does not affect the maximum supported rate.

Note that k^* is well defined if $S > U_N$, because (18) holds for $k = N$. The index k^* is the dividing point between peers with uplink capacities smaller than or equal to r_{max} and peers with uplink capacities larger than r_{max}. If $U_{k-1} > r \ge U_k$, then

$$\frac{1}{N}(S + \sum_{i=1}^{N} \hat{U}_i) = \frac{1}{N}(S + kr + \sum_{i=k+1}^{N} \hat{U}_i),$$

so we have $r = g(k)$. Now we give the detailed proof.

Proof. Case 1 is when the maximum upload rates of all nodes are affected by the single peer restriction. Case 2 is when none of the maximum upload rates are affected by the single peer restriction. Case 3 is when some maximum upload rates are affected while others are not. We combine cases 2 and 3 at the end.

Case 1: if $S \le U_N$, i.e., if the server uplink capacity is smaller than the smallest peer uplink capacity, then the streaming rate is limited by server uplink capacity, and we have $r_{max} = S$. This r_{max} is obviously achievable, as a simple chain topology ($s \to 1 \to 2 \to \cdots \to N$) works.

Case 2: if S is very large, such that a rate $r \ge U_1$ can be supported, then $\hat{U}_i = U_i, \forall i \in \mathcal{N}$, and thus the $M = 1$ constraint does not reduce the maximum upload rate of each user. Therefore, we can achieve the same maximum supported rate as in the unconstrained case (Equation (9)). The necessary condition for $r \ge U_1$ is

$$\min(S, \frac{1}{N}(S + \sum_{i=1}^{N} U_i)) \ge U_1 \Leftrightarrow S \ge \sum_{i=1}^{N} (U_1 - U_i) \text{ and } S \ge U_1, \tag{19}$$

which means that the rate $r = U_1$ can be supported.

Case 3: if $U_N < S < \max(U_1, \sum_{i=1}^{N}(U_1 - U_i))$, then U_N can be supported while U_1 cannot, and thus we have $U_1 > r_{max} > U_N$. Consider a given rate $r \in (U_N, U_1)$, suppose r lies between U_{k-1} and U_k, i.e., $U_{k-1} > r \ge U_k$. Then, $\hat{U}_i = r$, $\forall i < k$ and $\hat{U}_i = U_i$, $\forall i \ge k$. This means that the $M = 1$ constraint reduces the upload rate for peers 1 to $k-1$, but not for peers k to N. From (15), we have

$$r \le \min(S, \frac{1}{N}(S + \sum_{i=1}^{k-1} r + \sum_{i=k}^{N} U_i)) \Rightarrow r \le \min(S, g(k)). \tag{20}$$

Since $r \ge U_k$, we have $\min(S, g(k)) \ge U_k$, which means

$$S \ge U_k \text{ and } g(k) \ge U_k \Rightarrow S \ge \sum_{i=k}^{N} (U_k - U_i).$$

So if k satisfies (18), $\min(S, g(k))$ gives an upper bound for r. Now we look for the tightest such upper bound, which happens to be reached if $k = k^*$.

We have the following property for $g(k)$:

Lemma 2. *Let $g(k)$ be defined as in (17), then for any $k < N$,*

$$g(k) \ge U_k \Rightarrow g(k) \le g(k+1), \tag{21}$$

$$g(k) \le U_k \Rightarrow g(k) \ge g(k+1). \tag{22}$$

From Lemma 2, if k^* is the minimum k such that (18) holds, then $U_{k^*-1} > r_{max} \ge U_{k^*}$, and the tightest bound on r is

$$r_{max} \le \min(S, g(k^*)). \tag{23}$$

Under case 3, $U_1 > r$ and $k^* > 1$. However, if we remove the $U_1 > r$ condition, and allow S to be sufficiently large such that $k^* =$

1, then (18) reduces to (19), and thus we can think of case 2 as a special case of case 3 when $k^* = 1$.

The above analysis gives an upper bound for r_{max}. We already mentioned that a simple chain topology achieves the upper bound for case 1. For case 2 and 3, it is easy to see that the upper bound in (23) can be achieved by the same ring topology as illustrated in Figure 2, where $r = \min(S, g(k^*))$, and this ring topology can be derived by the same Bottleneck Removal multi-tree construction algorithm. □

Remark: Similar to the minimum server load problem, the multi-tree construction to achieve the maximum supported rate is either a chain or a ring topology and the multi-tree fanout for each user is 1, so we have $M_t = 1$. Therefore, the maximum supported rate under a stronger $M_t = 1$ constraint is also given by Equation (16).

4.3 Minimum Tree Depth for $M = 1$

In this subsection, we find the minimum multi-tree depth D_{min} that achieves the minimum server load s_{min} under the constraint $M = 1$. We first give a lower bound for D_{min}, and then show that the lower bound is achievable.

Suppose a multi-tree has L substream trees, and the server uploads to $H^{(l)}$ peers in substream tree l, which has rate $y^{(l)}$, $l = 1, 2, \cdots, L$. We say there are $H^{(l)}$ *branches* in this tree. After receiving the substream from the server, these $H^{(l)}$ peers further upload to the other peers. Since each peer has at most one child, there are $H^{(l)}$ leaf nodes and $N - H^{(l)}$ non-leaf nodes in tree l. Recall that the set of leaf nodes is $E^{(l)}$ and the set of non-leaf nodes is $F^{(l)}$.

The depth of tree l, denoted by $D^{(l)}$, is lower bounded by $\lceil N/H^{(l)} \rceil$, since there exists a branch containing at least $\lceil N/H^{(l)} \rceil$ nodes, and in this branch the nodes form a chain. This lower bound is tight, since evenly distributing all nodes among the $H^{(l)}$ branches achieves it.

The depth of the multi-tree, denoted by D, is lower bounded as in the following equation:

$$D = \max_l D^{(l)} \geq \max_l \left\lceil \frac{N}{H^{(l)}} \right\rceil = \left\lceil \frac{N}{\min_l H^{(l)}} \right\rceil, \qquad (24)$$

and by evenly distributing nodes among branches in any tree, this bound is also tight. Note that for simplicity of notation, throughout the paper, $\max_l$ or $\min_l$ represents the maximization or minimization over all substream trees $l = 1, 2, \cdots, L$ in a given multi-tree. From (24), in order to minimize D, we need to minimize the largest $D^{(l)}$, or equivalently, maximize $\min_l H^{(l)}$.

We first give an upper bound on $\min_l H^{(l)}$ and thus a lower bound on D. Suppose we construct a set of substream trees to support r with minimum server load s_{min}, we have

$$\begin{aligned} r &= \sum_{l=1}^{L} y^{(l)}, \\ s_{min} &= \sum_{l=1}^{L} H^{(l)} y^{(l)}. \end{aligned}$$

So

$$s_{min} \geq \min_l H^{(l)} \sum_{l=1}^{L} y^{(l)} = \min_l H^{(l)} r,$$

and

$$\min_l H^{(l)} \leq \lfloor \frac{s_{min}}{r} \rfloor.$$

Therefore we have the following lower bound on multi-tree depth:

$$D \geq \left\lceil \frac{N}{\min_l H^{(l)}} \right\rceil \geq \left\lceil \frac{N}{\lfloor \frac{s_{min}}{r} \rfloor} \right\rceil. \qquad (25)$$

This gives a lower bound on D for any multi-tree depth. The bound is in fact achievable, so we have the following theorem:

Theorem 4. *For a given streaming rate r, under the $M = 1$ constraint, the minimum depth among all multi-trees that achieves the minimum server load s_{min} is*

$$D_{min} = \left\lceil \frac{N}{\lfloor \frac{s_{min}}{r} \rfloor} \right\rceil. \qquad (26)$$

Proof. From (25), we have $D \geq D_{min}$, where D_{min} is defined in (26). We now prove that $D = D_{min}$ is achievable. Recall that $s_{min} = \max(r, \sum_{i=1}^{N} (r - U_i)^+)$. We consider two cases.

Case 1: $s_{min} < 2r$. There is a substream tree with only one branch, so $D_{min} = N$. The tree depth of N can be achieved with the ring topology in Figure 2.

Case 2: $s_{min} \geq 2r$. It is sufficient to construct a multi-tree such that the streaming rate r is supported by s_{min} and in each substream tree $l = 1, 2, \cdots, L$, $H^{(l)} \geq \lfloor \frac{s_{min}}{r} \rfloor$.

Consider tree l with $H^{(l)}$ branches. The $N - H^{(l)}$ non-leaf nodes upload to peers at rate $y^{(l)}$, and the $H^{(l)}$ leaf nodes do not upload. Since each branch is a chain, once the leaf nodes are fixed, the non-leaf nodes can be evenly distributed among the branches so that the depth of tree l is $\lceil \frac{N}{H^{(l)}} \rceil$. This indicates that constructing a substream tree with $H^{(l)}$ branches is equivalent to selecting $H^{(l)}$ leaf nodes and subtracting $y^{(l)}$ from the other $N - H^{(l)}$ nodes' capacities.

Since the server load is larger than r, from Section 4.1 we must have that peer i uploads at $\hat{U}_i = \min(r, U_i)$. For a given rate r, let k be the smallest index such that $U_k < r$, i.e.,

$$U_1 \geq U_2 \geq \cdots \geq U_{k-1} \geq r > U_k \geq \cdots \geq U_N.$$

Note that we must have $U_N < r$; otherwise we would have $s_{min} = \max(r, \sum_{i=1}^{N} (r - U_i)^+) = r$, contradicting $s_{min} \geq 2r$. Then the nodes 1, 2, $\cdots, k-1$ upload at rate r and the nodes $k, k+1, \cdots, N$ upload at rates less than r. So only the nodes $k, k+1, \cdots, N$ can be leaf nodes in substream trees and we only need to consider them when constructing the trees. Therefore without loss of generality, we assume $r > U_1$.

Now we need an algorithm to construct the trees. Let

$$H_1 := \lfloor \frac{s_{min}}{r} \rfloor, \; J := N - \lfloor \frac{s_{min}}{r} \rfloor. \qquad (27)$$

Then J is the maximum number of non-leaf nodes whose uplink capacities will be used in a tree. We reduce the uplink capacity of these nodes and the remaining is called residual capacity. The idea is to equalize the residual capacities of the nodes so that in the end they can all be consumed. We start with a particular node in the middle and make more and more adjacent nodes have the same residual capacity. Since the group grows like a snowball, we call this the Snowball Algorithm.

The Snowball Algorithm for Tree Construction:

We first introduce some notations on multi-tree construction. Suppose we construct the multi-tree step by step, and by slightly abusing notation, we use l to index the steps. In each step, there may be one, or multiple substream trees constructed, but within one step, the branch number in any substream tree is the same, denoted by $H^{(l)}$. We use $y^{(l)}$ to denote the supported rate in step l; use $U_i^{(l)}$ to denote the residual capacity for peer i, $i = 1, 2, \cdots, N$ after the l-th step; use $\delta^{(l)}$, $s^{(l)}$, and $r^{(l)}$ to denote the total supported rate, the residual server load, and the remaining streaming rate to support, respectively, after the l-th step. Initially, for $l = 0$, we define $\delta^{(0)} = 0$, $U_i^{(0)} = U_i, i = 1, 2, \cdots, N$, $s^{(0)} = s_{min}$, and $r^{(0)} = r$.

Then we have the following equations:

$$\begin{aligned}
\delta^{(l)} &= \delta^{(l-1)} + y^{(l)}, \\
s^{(l)} &= s^{(l-1)} - y^{(l)} H^{(l)}, \\
r^{(l)} &= r^{(l-1)} - y^{(l)}, \\
U_i^{(l)} &= \begin{cases} U_i^{(l-1)} & \text{if } i \in E^{(l)}, \\ U_i^{(l-1)} - y^{(l)} & \text{if } i \in F^{(l)}, \end{cases} \\
y^{(l+1)} &\leq \min(\frac{s}{H^{(l)}}, U_i^{(l)}, i = 1, 2, \cdots, N).
\end{aligned}$$

After L steps in a successful multi-tree construction (by successful, we mean that the streaming rate r is supported by the minimum server load), we have the final stage, at which $s^{(L)} = r^{(L)} = 0$, and $U_i^{(l)} = 0$, $\forall i = 1, 2, \cdots, N$.

With the above notations at hand, we introduce the following algorithm on the multi-tree construction: at step 1, we trim node 1 to J, i.e., nodes $J+1$ to N are leaf nodes, such that $U_J^{(1)} = U_{J+1}^{(1)}$. Here, by "trim" we mean "reduce the residual uplink capacity of the node". This is easily done as we can set $y^{(1)} = U_J - U_{J+1}$. At step 2, we construct two trees with the same rate, one trims nodes 1 to J and the other trims node 1 to $J-1$ and $J+1$, and the trimming stops when one of the two following cases occurs: either

$$U_{J-1}^{(2)} = U_J^{(2)} = U_{J+1}^{(2)},$$

or

$$U_J^{(2)} = U_{J+1}^{(2)} = U_{J+2}^{(2)}.$$

The first case occurs if

$$U_{J-1}^{(1)} - U_J^{(1)} \leq U_{J+1}^{(1)} - U_{J+2}^{(1)},$$

and the second case occurs otherwise. In both cases, after step 2, including the J-th node, there are at least 3 nodes that have the same residual capacity. We use the term *snowball* to represent the set of nodes whose residual capacities are the same as that of the J-th node. Let similar trimming continue in latter steps: suppose before the l-th step, the snowball contains m_1 nodes, and there are m_2 nodes in front of the snowball and their residual capacities are larger than that of the snowball. Since J-th node is in the snowball, $m_1 + m_2 \geq J$. In the l-th step, m_1 substream trees with the same rate are constructed, where each tree trims J nodes. The m_2 nodes in front of the snowball are trimmed in each tree, while each node inside the snowball is trimmed in $J - m_2 < m_1$ trees. This way, the nodes in front of the snowball are trimmed faster than the nodes inside the snowball, and the nodes after the snowball are not trimmed at all. The l-th step stops if the node just before or just after the snowball is absorbed into the snowball.

Two examples of this algorithm are given in Figure 4. A feature of the Snowball algorithm is that the order of the residual capacities of all nodes at any stage is the same as the order of the original capacities, i.e.,

$$U_1^{(l)} \geq U_2^{(l)} \geq \cdots \geq U_N^{(l)}.$$

After a sufficient number of steps, say K steps, the Snowball algorithm stops at one of the following: either

$$U_1^{(K)} = U_2^{(K)} = \cdots = U_N^{(K)} = u,$$

or

$$U_{J'+1}^{(K)} = U_{J'+1}^{(K)} = \cdots = U_N^{(K)} = 0 \text{ for some } J' < J.$$

In the first case, the snowball finally contains all the nodes; and in the second case, before the snowball contains all the nodes, all the nodes in the snowball have zero residual capacity. Since the J-th node is in the snowball, in the second case, only $J' < J$ nodes have positive residual capacities.

Example 1: $N = 5, r = 11, s = 34, H_1 = 3, J = 2, q = 0$.

Step (l)	$U_1^{(l)}$	$U_2^{(l)}$	$U_3^{(l)}$	$U_4^{(l)}$	$U_5^{(l)}$	$\delta^{(l)}$	$r^{(l)}$	$s^{(l)}$
0	10	5	(3)	2	1	0	11	34
1	8	(3	3)	2	1	2	9	28
2	6	(2	2	2)	1	4	7	22
3	3	(1	1	1	1)	7	4	13
4	($\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$)	$\frac{29}{3}$	$\frac{4}{3}$	5
5	(0	0	0	0	0)	$\frac{21}{2}$	$\frac{1}{2}$	$\frac{5}{2}$
6	(0	0	0	0	0)	11	0	0

Example 2: $N = 5, r = 11, s = 30, H_1 = 2, J = 3, q = 8$.

Step (l)	$U_1^{(l)}$	$U_2^{(l)}$	$U_3^{(l)}$	$U_4^{(l)}$	$U_5^{(l)}$	$\delta^{(l)}$	$r^{(l)}$	$s^{(l)}$
0	10	9	3	(2)	1	0	11	30
1	9	9	(2	2)	1	1	10	28
2	7	6	(1	1	1)	3	8	24
3	4	3	(0	0	0)	6	5	18
4	1	0	(0	0	0)	9	2	9
5	0	0	(0	0	0)	10	1	5
6	0	0	(0	0	0)	11	0	0

Figure 4: Two examples for the Snowball algorithm. In the tables, $(\cdot)$ indicates the snowball. In example 1, the snowball finally includes all nodes, and the multi-tree construction is successful: it achieves the minimum server load $s_{min} = 34$, and its minimum branch number is $H_1 = 3$. In example 2, all nodes have zero residual capacity before the snowball includes all nodes, and the multi-tree construction is successful: it achieves the minimum server load $s_{min} = 30$, and its minimum branch number is $H_1 = 2$.

After the snowball algorithm stops, in both cases, we can further construct trees with at least H_1 branches, i.e., with at most J nodes trimmed, and we can prove that we can achieve both the minimum server load and the $H_{min} = H_1$ property for both cases. That result is summarized in the following lemma:

Lemma 3. *Using the Snowball algorithm, we can construct a multi-tree such that the minimum server load is achieved, and the branch number is at least H_1 in any substream tree.*

From the proof in the Appendix, the algorithm runs in polynomial time. Lemma 3 completes the proof of Theorem 4. □

We have found the lower bound on D_{min} for $M = 1$. If we let the server upload rate be bigger than s_{min}, then following similar steps, we can show that an equation similar to (26) holds:

$$D_{min} = \left\lceil \frac{N}{\lfloor \frac{s}{r} \rfloor} \right\rceil. \tag{28}$$

Equation (28) gives a tradeoff between D_{min} and s, or the delay and the server load. This means in real systems, by increasing the server capacity, we can achieve a smaller tree depth and a lower delay.

4.4 Minimum Tree Depth for $M_t = 1$

We now study the minimum tree depth under the $M_t = 1$ constraint. We show that the best strategy to reduce tree depth under the $M_t = 1$ constraint is to divide the original ring (see Figure 2) into multiple rings while maintaining the minimum server load. Since

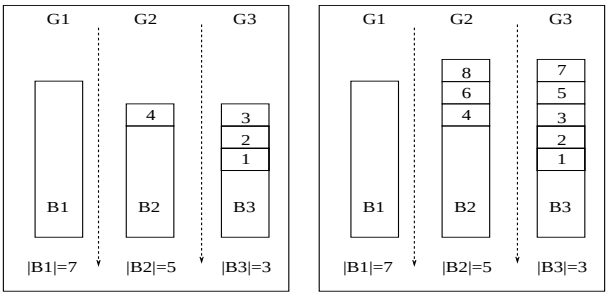

Figure 5: An example to show the waterfilling strategy. Here, $H=3$, $|B_1|=7, |B_2|=5, |B_3|=3$, $A=\{1,2,\cdots,k-1\}$, and the height of each bin represents the number of elements in the bin. One-by-one, the elements of A are put into the bin with the least element (waterfilling). If $|A|<6$, as in the left figure, then the three bins are not flattened, and $\max(|G_h|, h=1,2,3)=|B_1|$. If $|A|\geq 6$, a in the right figure, then the three bins are not flattened, and $\max(|G_h|, h=1,2,3)=\lceil N/H\rceil$.

the tree depth of a ring is the number of nodes in this ring, by dividing the N nodes into multiple rings, the multi-tree depth is the maximum ring size. And to achieve a minimum tree depth, we need to minimize the maximum ring size.

Before we give the minimum value of maximum ring size, we first introduce further notation and terminology. Any subset $P\subset\mathcal{N}$ is an index set, and we denote the number of its element by $|P|$. We say P is *isolated* if no node in P uploads to or downloads from any nodes outside P. If the nodes in P form a ring topology, then the depth of P is $|P|$. For an integer $H\leq|P|$, we call a set of index sets $\{P_1,P_2,\cdots,P_H\}$ a *division*, or an *H-division*, of P, if all P_i, $i=1,2,\cdots,H$, are mutually exclusive and their union equals P.

We define the two sets A and B: A contains the nodes whose capacities are at least r, and B contains the rest. Suppose $A=\{1,2,\cdots,k-1\}$ and $B=\{k,k+1,\cdots,N\}$, then $|A|=k-1$ and $|B|=N-k+1$. Consider an H-division of $\mathcal{N}$ into $\{G_1, G_2, \cdots, G_H\}$, and define $A_h=G_h\cap A$ and $B_h=G_h\cap B$, then $\{A_h, h=1,2,\cdots,H\}$ is an H-division of A and $\{B_h, h=1,2,\cdots,H\}$ is an H-division of B. If the H sets $G_1, G_2, \cdots, G_H$ are isolated, i.e., no cross set uploading/downloading, then we call $\{G_1,G_2,\cdots,G_H\}$ an *isolated H-division* of $\mathcal{N}$. We call an isolated H-division of $\mathcal{N}$ to be *efficient*, if the minimum server load is not affected by making $G_1,G_2,\cdots,G_H$ isolated. If the nodes in each G_h connect to be a ring, by efficiently dividing $\mathcal{N}$ into $G_1,G_2,\cdots,G_H$, the multi-tree depth is reduced from N to $\max(|G_h|, h=1,2,\cdots,H)$.

The condition for an isolated division to be efficient is given in the following lemma:

Lemma 4. *The necessary and sufficient condition for an isolated H-division to be efficient is*

$$\sum_{i\in B_h}(r-U_i)\geq r,\ \forall h\in 1,2,\cdots,H\,. \tag{29}$$

Note that for an efficient H-division, there are only requirements on $B_h, h=1,2,\cdots,H$, and no requirements on $A_h, h=1,2,\cdots,H$. This means that once we divide B into H bins such that (29) is satisfied, we can arbitrarily put the nodes in A into the H bins. Given an efficient H-division of B, the best strategy of putting A nodes into the bins is the *waterfilling* strategy, illustrated in Figure 5, which minimizes $\max(|G_h|, h=1,2,\cdots,H)$. From the waterfilling strategy, for any fixed efficient division of B into $\{B_h, h=1,2,\cdots,H\}$, without loss of generality, suppose B_1 has the most elements, i.e., $|B_1|=\max_h|B_h|$, then the minimum multi-tree depth is

$$D_w=\begin{cases}\lceil N/H\rceil & \text{if } |A|\geq\sum_{h=2}^{H}(|B_1|-|B_h|)\,,\\ \max(|B_h|, h=1,2,\cdots,H) & \textit{otherwise}\,,\end{cases} \tag{30}$$

where the subscript w stands for waterfilling. The first case corresponds to the scenario that the H bins are filled to be flat, and the second corresponds to the scenario that the bins are not flattened so the highest original bin gives the largest height of the filled bins.

The problem of finding the minimum D_w is strongly-NP hard, because the first case, i.e., when $|A|\geq\sum_{h=2}^{H}(|B_1|-|B_h|)$, is strongly-NP hard. In the first case, minimizing D_w is equivalent to maximizing the number of bins H, such that in each bin h, $\sum_{i\in B_h}(r-U_i)\geq r$. This is the classic bin covering problem, or the dual bin packing problem [9, 10, 11], which is known to be strongly-NP hard [12]. There are polynomial time approximation algorithms [13, 14, 15].

Suppose we can find $\min D_w$, then we get a upper bound on D_{min}. Since the set of multi-trees under the $M_t=1$ constraint is a subset of that under the $M=1$ constraint, from (26), we know that the $\lceil\frac{N}{\lfloor\frac{s_{min}}{r}\rfloor}\rceil$ lower bounds D_{min}. Then, we have the following theorem:

Theorem 5. *For a given streaming rate r, under the $M_t=1$ constraint, the minimum depth among all multi-trees that achieve the minimum server load s_{min} satisfies the following bounds:*

$$\left\lceil\frac{N}{\left\lfloor\frac{s_{min}}{r}\right\rfloor}\right\rceil\leq D_{min}\leq\min D_w\,, \tag{31}$$

where the minimum of D_w is over all efficient divisions.

Lemma 4 is the condition for achieving minimum server load. If the server uploads at a rate higher than s_{min}, then the condition in Lemma 4 can be relaxed, and not all divisions need to satisfy (29). This way we may be able to have more divisions and therefore reduce the tree depth. This is a tradeoff between server load and delay.

5. MULTIPLE DOWNLOADING PEERS

We have studied the two extreme cases in Sections 3 and 4, where the peer selection has the most and least freedom, respectively. In this section, we study the general case.

5.1 Homogeneous Users: Maximum Streaming Rate and Minimum Server Upload

We start with a homogeneous users scenario, where $U_1=U_2=\cdots=U_N=U$. Perhaps surprisingly, when all the peers have the same uplink capacity, the peer number constraint does not change the minimum server load and the maximum supported rate.

Theorem 6. *When user uplink capacities are homogeneous, the minimum server load and maximum supported rate are the same as in the unconstrained case, independent of M or M_t.*

Proof. The unconstrained peer selection case gives performance bounds for constrained peer selection. We show that these bounds are achievable in the homogeneous uplink case even with the most strict peer selection constraint: $M_t=1$. Therefore the bounds must be achievable for all M and M_t, because the multi-tree for $M_t=1$ is also a valid multi-tree for general M and M_t values.

When peers have the same uplink capacity, Theorem 2 reduces to

$$s_{min}(r)=\max(r, N(r-U))\,, \tag{32}$$

and Theorem 3 reduces to

$$r_{max}(S) = \min(S, \frac{S+NU}{N}). \quad (33)$$

These values for $M_t = 1$ is the same as the values for unconstrained peer selection. □

5.2 Homogeneous Users: Minimum Tree Depth

We have seen in Section 4 that the biggest effect of restricting peer selection is the increase in tree depth. We now study the tree depth for general $1 \le M \le N-1$ and homogeneous uplinks. Sometimes the exact value of the minimum tree depth cannot be given analytically, and we give upper and lower bounds that are very close to each other.

The server has no limit on the number of connections, but a user can upload to at most M peers, so a substream tree is a number of M-ary trees with the roots connected to the server. In order to minimize the tree depth, we need to divide the users into as many groups as possible where each group forms an M-ary tree without increasing the server load. We give upper and lower bounds on the maximum number of groups: H_{max} and H_{min}, and the maximum number of peers in a group is bounded by

$$N_{min} := \left\lceil \frac{N}{H_{max}} \right\rceil, \quad (34)$$

and

$$N_{max} := \left\lceil \frac{N}{H_{min}} \right\rceil. \quad (35)$$

Since all the users are identical, we can build different substream trees by rotating the users within each group. So the substream trees all have the same depth, which is also the multi-tree depth. The following lemma gives the tree depth:

Lemma 5. *If a group of n homogeneous users form an M-ary tree with the root connected to the server, then the minimum tree depth is given by*

$$f(n,M) := \begin{cases} n & \text{if } M = 1, \\ \lceil log_M((M-1)n+1) \rceil & \text{if } M > 1. \end{cases}$$

When different substream trees are formed by rotating the users in the M-ary tree, then the minimum multi-tree fanout M_t has the following upper bound:

$$\min M_t \le M + (f(n,M)-2)(M-1),$$

We have the following theorem for the minimum tree depth with identical users:

Theorem 7. *Suppose one user has at most M children in any substream tree, and that all users have the same uplink capacity U. Then, the minimum depth that achieves the minimum server load is*

1. *If $N \le 2$, then $D_{min} = N$.*
2. *If $N > 2$ and $r < \frac{NU}{N-2}$, then $D_{min} = f(N,M)$, and*
$$\min M_t \le M + (f(N,M)-2)(M-1).$$
3. *If $N > 2$ and $r \ge \frac{NU}{N-2}$, then*
$$f(N_{min},M) \le D_{min} \le f(N_{max},M),$$
and
$$\min M_t \le M + (f(N_{max},M)-2)(M-1),$$

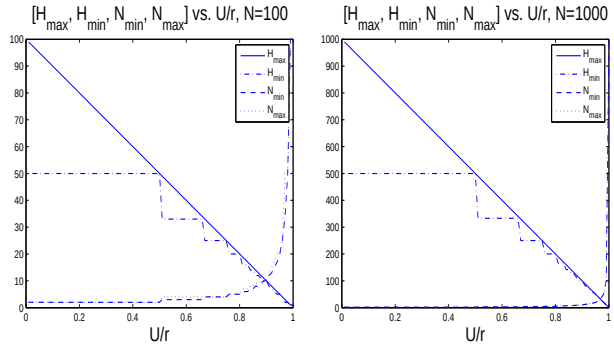

Figure 6: The values of $H_{max}, H_{min}, N_{min}$ and N_{max} versus U/r for $N = 100$ and $N = 300$. We can see that N_{min} and N_{max} are close to each other which means the bounds in Theorem 7 are tight.

where N_{min} and N_{max} are defined in (34) and (35), and

$$H_{max} = \left\lfloor N\left(\frac{r-U}{r}\right) \right\rfloor,$$
$$H_{min} = \left\lfloor \frac{N}{\lceil \frac{r}{r-U} \rceil} \right\rfloor.$$

Proof. We prove this theorem under the three different cases. In the first two cases we cannot divide the N nodes into groups; in case 3 we can.

Case 1: If $N = 1$, then $D_{min} = 1$. If $N = 2$, then $D_{min} = 2$. For both, $D_{min} = N$.

Case 2: When $N > 2$ and $r < NU/(N-2)$, we have

$$s_{min} = \max(r, N(r-U)) < 2r.$$

So there exists at least one substream tree in which the server uploads to only one user, i.e., there is only one branch; otherwise, if the server uploads every substream at least twice, we would have $s \ge 2r$ and there is a contradiction. From Lemma 5 the minimum depth is $f(N,M)$.

Case 3: We first check the lower bound. If $N(r-U) \ge r$, then there is a substream that the server transmits to at most $\lfloor N(r-U)/r \rfloor$ users. This means that there is a substream that the server transmits to at most H_{max} users. Therefore at least one user who receives the substream from the server is responsible for passing it (possibly through multiple hops) to at least $N_{min}-1$ other users. From Lemma 5, we have $D_{min} \ge f(N_{min},M)$.

We then look at the upper bound. We have

$$\left\lfloor \frac{N}{H_{min}} \right\rfloor (r-U) \ge \left\lceil \frac{r}{r-U} \right\rceil (r-U) \ge r.$$

Therefore we can divide the users into H_{min} groups without increasing the server load. From Lemma 5, we have $D_{min} \le f(N_{max},M)$. □

Theorem 7 gives an upper bound and a lower bound, rather than the exact expression, on D_{min} for case 3. We check numerically whether the bounds are tight and Figure 6 shows the result. For $N = 100$ and $N = 300$, we vary U/r from 0 to 1. The plots show that the bounds are very tight: N_{min} and N_{max} are close to each other, and they grow closer for larger N. This means $f(N_{min},M)$ and $f(N_{max},M)$ are close to each other.

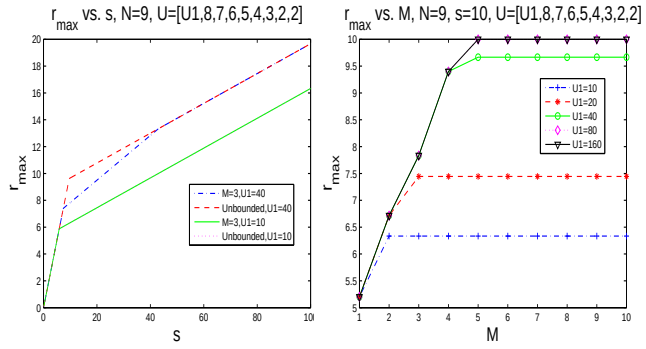

Figure 7: The maximum supported rate as functions of server load (left figure) and degree bound (right figure). In both figures, we have $N = 9$ users, fix the upload bandwidths for users 2 to 9, and vary user 1's bandwidth. The last two curves overlap in both plots.

In Section 4, we already saw the tradeoff between server load and minimum tree depth for $M = 1$ and $\hat{M} = 1$. Theorem 7 further shows the tradeoff between M and the minimum tree depth.

5.3 Heterogeneous Users

We have considered single downloading peer and heterogeneous users case in Section 4.1. Similar to (12) and (13), for general $M > 1$, the maximum upload rate of user i is

$$\hat{U}_i(M) := \min(Mr, U_i), \tag{36}$$

and the lower bound for $s_{min}(r,M)$ under $M > 1$ degree bound is:

$$\begin{aligned} s_{min}(r,M) &\geq \max\left(r, Nr - \sum_{i=1}^{N} \hat{U}_i(M)\right) \\ &= \max\left(r, Nr - \sum_{i=1}^{N} \min(Mr, U_i)\right). \end{aligned} \tag{37}$$

It is shown in [16] that this bound is in fact tight.

Similar to Theorem 3 in Section 4.2, given the minimum server load, we have the following result on maximum supported rate:

$$r_{max}(S,M) = \begin{cases} S & \text{if } S \leq \frac{U_N}{M}, \\ \min(S, g(k^*,M)) & \text{if } S > \frac{U_N}{M}, \end{cases} \tag{38}$$

where

$$g(k,M) := \begin{cases} \frac{S+\sum_{i=k}^{N} U_i}{N-Mk+M} & \text{if } k < \frac{N}{M}+1, \\ \infty & \text{otherwise,} \end{cases} \tag{39}$$

and k^* is the minimum k such that

$$U_k \leq Mg(k,M) \quad \text{and} \quad U_k \leq MS. \tag{40}$$

Given the minimum server load, similar to Theorem 4, we can also show that, for a given streaming rate r, under the degree bound $M > 1$, the minimum depth among all trees that achieve the minimum server load $s_{min}(r,M)$ is

$$f(N',M) \leq D_{min} \leq f(N,M), \tag{41}$$

where $N' := \left\lceil \frac{N}{\lfloor \frac{s_{min}(r,M)}{r} \rfloor} \right\rceil$. The proofs of the maximum streaming rate and the bounds on the minimum tree depth are available in [17].

Some numerical examples on the maximum supported rate as functions of server load, user uplink bandwidths, and degree bounds are plotted in Figure 7. The figure on the left is the maximum streaming rate as a function of server capacity, under different tree out-degree bounds and user uplink capacities. This plot shows that when user uplink capacities are close, the tree out-degree bound has little effect. The bound has effect when the user uplink capacities differ significantly. This is because the constraint on peer selection causes some high user uplink capacity to be under-utilized. The figure on the right is the maximum streaming rate as a function of tree out-degree bound, for different user uplink capacities. We observe that in each case, there is a value of M that if the degree bound is increased beyond this value, there is no gain in maximum supported streaming rate. This is because when enough peers download from high capacity users to fully utilize their uplink capacities, allowing more connections does not help the maximum streaming rate. In addition, when the maximum streaming rate is limited by the server capacity, increasing user uplink capacity or the degree bound cannot help.

6. RELATED WORK

The concept of peer-assisted live streaming has received a lot of attention in recent years. The application scenarios include IPTV [18, 19, 20, 21, 22], video-on-demand [23, 24, 5, 25], and video conferencing [26]. Our effort focuses on the analysis of large-scale IPTV systems. Peer-assisted IPTV systems can be tree-based [27, 28, 7], where the tree structure is constructed and maintained centrally and explicitly, mesh-based [29, 30], where each peer selects partners to trade packets like BitTorrent and the system does not explicitly construct and maintain a tree structure, or tree-mesh hybrid-based [31]. For a comparison of tree and mesh, see [32]; for an overview of challenges and approaches in current large-scale P2P streaming, see [2].

While there is much work on system design and measurement studies of a peer-to-peer streaming system, few papers work on theoretical analysis and fundamental limitations of peer-assisted live streaming system. In [6, 7], the maximum supported rate without degree bound is studied; [6] further develops a stochastic fluid model to consider peer churn; and [26] extends the unconstrained performance study to video conferencing scenario. In this paper, we first bring peer selection constraint or outgoing degree bound into the framework, and derive the optimal performance under different peer selection constraints by first giving the performance bounds and then constructing multi-trees to achieve the bounds. Note that although the bounds are achieved by tree-based algorithms, they also apply to mesh-based algorithms. Therefore, these bounds not only are optimal values for tree-based algorithms, but also can server as a benchmark to compare mesh-based algorithms.

7. CONCLUSION AND FUTURE WORK

In this paper, we have studied three performance metrics: minimum server load, maximum supported rate, and minimum tree depth, under three cases: unconstrained peer selection, single peer selection, and constrained peer selection. We derive bounds on these metrics and prove the bounds are tight. The analysis on the performance bounds also suggest the tradeoffs between tree depth, server load, and degree bound.

There are several directions to extend this work. First, we consider a single streaming rate in this paper, and we can extend the discussion to multi-layer streams, where users with different downlink capacities receive different number of layers of the stream. Second, we consider a single session, and we can extend to multi-session conferential scenario. Finally, we can study more on the

tradeoffs between tree depth, server load and degree bound, like exploring the 3-D tradeoff region.

8. ACKNOWLEDGMENT

We thank Jiayue He, Professor Gary Chan, and Dr. David Johnson for their useful comments.

9. REFERENCES

[1] Y.-C. Tu, J. Sun, M. Hefeeda, and S. Prabhakar, "An analytical study of peer-to-peer media streaming systems," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, no. 4, pp. 354–376, 2005.

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

[3] T. Silverston and O. Fourmaux, "Measuring P2P IPTV Systems," in *Proceedings of the 17th International workshop on Network and Operating Systems Support for Digital Audio & Video*, 2007.

[4] P. Rodriguez, S.-M. Tan, and C. Gkantsidis, "On the Feasibility of Commercial, Legal P2P Content Distribution," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 75–78, 2006.

[5] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Valleo, "Push-to-Peer Video-on-Demand system: Design and Evaluation," *IEEE Journal on Selected Areas in Communications, special issue on Advances in Peer-to-Peer Streaming Systems*, 2007.

[6] R. Kumar, Y. Liu, and K. Ross, "Stochastic Fluid Theory for P2P Streaming Systems," in *Infocom 2007*, (Anchorage, Alaska), 2007.

[7] J. Li, P. A. Chou, and C. Zhang, "Mutualcast: An efficient mechanism for one-to-many content distribution," in *ACM SIGCOMM ASIA Workshop*, 2005.

[8] B. Cohen, "Incentives Build Robustness in BitTorrent," in *First Workshop on the Economics of Peer-to-Peer Systems*,, 2003.

[9] S. Assmann, D. Johnson, D. Kleitman, and J. Leung, "On a Dual Version of the One-Dimensional Bin Packing Problem," *Journal of Algorithms*, vol. 5, pp. 502–525, 1984.

[10] S. Assmann, "Problems in Discrete Applied Mathematics," in *PhD Thesis*, (Massachusetts Institute of Technology, Cambridge, MA), 1983.

[11] J. Csirik and J. Frenk, "A dual version of bin packing," Tech. Rep. 9029-a, Erasmus University of Rotterdam - Econometric Institute, 1990.

[12] M. R. Garey and D. S. Johnson, ""Strong" NP-Completeness Results: Motivation, Examples, and Implications," *J. ACM*, vol. 25, no. 3, pp. 499–508, 1978.

[13] J. Csirik, J. B. G. Frenk, M. Labbé, and S. Zhang, "Two Simple Algorithms for Bin Covering," *Acta Cybern.*, vol. 14, no. 1, pp. 13–25, 1999.

[14] J. Csirik and V. Totik, "Online Algorithms for a Dual Version of Bin Packing," *Discrete Appl. Math.*, vol. 21, no. 2, pp. 163–167, 1988.

[15] J. Csirik and G. J. Woeginger, "On-line Packing and Covering Problems," in *Developments from a June 1996 seminar on Online algorithms*, (London, UK), pp. 147–177, Springer-Verlag, 1998.

[16] S. Liu, S. Sengupta, M. Chiang, J. Li, and P. A. Chou, "Achieving streaming capacity in P2P." Microsoft Research Technical Report, April 2008.

[17] S. Liu, M. Chiang, S. Sengupta, J. Li, and P. A. Chou, "Streaming capacity for p2p with degree bound," in *Annual Allerton Conference on Communication, Control, and Computing*, September 2008.

[18] G. Dan, V. Fodor, and I. Chatzidrossos, "On the Performance of Multiple-Tree-Based Peer-to-Peer Live Streaming," *IEEE INFOCOM*, pp. 2556–2560, May 2007.

[19] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System," in *Proc. of IPTV Workshop, International World Wide Web Conference*, 2006.

[20] S. Ali, A. Mathur, and H. Zhang, "Measurement of commercial peer-to-peer live video streaming," in *Proceedings of the Workshop in Recent Advances in Peer-to-Peer Streaming (WRAIPS)*, 2006.

[21] T. Small, B. Liang, and B. Li, "Scaling laws and tradeoffs in peer-to-peer live multimedia streaming," in *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, (New York, NY, USA), pp. 539–548, ACM, 2006.

[22] T. Locher, R. Meier, S. Schmid, and R. Wattenhofer, "Push-to-Pull Peer-to-Peer Live Streaming," in *21st International Symposium on Distributed Computing (DISC)*, September 2007.

[23] C. Huang, J. Li, and K. Ross, "Peer-Assisted VoD: Making Internet Video Distribution Cheap." IPTPS'07, Redmond, 2007.

[24] C. Huang, J. Li, and K. W. Ross, "Can Internet Video-on-Demand be Profitable?," in *ACM SIGCOMM*, (New York, NY, USA), pp. 133–144, ACM, 2007.

[25] T. Piotrowski, S. Banerjee, S. Bhatnagar, S. Ganguly, and R. Izmailov, "Peer-to-Peer Streaming of Stored Media: The Indirect Approach," *SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 1, pp. 371–372, 2006.

[26] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility maximization in peer-to-peer systems," in *ACM Sigmetrics*, 2008.

[27] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *ACM NOSSDAV*, (Miami Beach, FL), May 2002.

[28] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in a cooperative environment," in *ACM SOSP'03*, (Lake Bolton, New York), October 2003.

[29] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "DONet/CoolStreaming: A data-driven overlay network for live media streaming," in *IEEE INFOCOM*, (Miami, FL), March 2005.

[30] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," *IEEE Journal on Selected Areas in Communications, Special Issue on Advances in Peer-to-Peer Streaming Systems*, vol. 25, pp. 1655–1666, December 2007.

[31] F. Wang, Y. Xiong, and J. Liu, "mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast," in *the 27th International Conference on Distributed Computing Systems table of contents*, 2007.

[32] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree:

A comparative study of live P2P streaming approaches," in *IEEE Infocom*, (Anchorage, AK), pp. 1424–1432, May 2007.

10. APPENDIX

In the appendix, we prove all the lemmas in the paper.

10.1 Proof of Lemma 1

We first give the following result.

Lemma 6. *The Bottleneck Removal algorithm preserves the following invariant after each iteration:*

$$u_i^{(l)} + s_{i+1}^{(l)} = c^{(l)}, \quad \forall i = 1,2,\cdots,N, \quad \forall l = 0,1,2,\cdots$$

Proof. We prove the lemma by induction. When $l = 0$, $u_i^{(l)} + s_{i+1}^{(l)} = r$, as indicated by the streaming scheme.

Suppose the claim holds for $l = t$, e.g., $u_i^{(t)} + s_{i+1}^{(t)} = c^{(t)}$. When $l = t+1$, the change of residual capacities occurs only on those edges which are on the tree T_t. Consider a user i, either $(i, i+1) \in T_t$, or $(s, i+1) \in T_t$. As a result, either $u_i^{(t+1)} = u_i^{(t)} - y_t$ or $s_{i+1}^{(t+1)} = s_{(i+1)}^{(t)} - y_t$. By induction, we have $u_i^{(t+1)} + s_{i+1}^{(t+1)} = c^{(t)} - y_t = c^{(t+1)}$. □

From Lemma 6, we have the following corollary:

Corollary 1. *During each iteration of the Bottleneck Removal algorithm, there exists k such that either $u_k^{(l)}$ becomes zero, or $s_{k+1}^{(l)}$ becomes zero.*

Proof. Given the way k is chosen in step (3) of the Bottleneck Removal algorithm (Table 3), we know that k is the starting point of the bottleneck edge, whose residual capacity will be reduced to zero in step (14). □

We also need the following lemma:

Lemma 7. *The Bottleneck Removal algorithm constructs a tree which spans all N users in each iteration.*

Proof. The essence of the proof is to show that step (6) and (11) in the algorithm actually builds a tree branch which sustains a streaming rate $y^{(l)}$ given in step (5).

We first show step (6) is correct. Consider any user p' between p and k on the path $p \to \cdots \to p' \to \cdots \to k$. The choice of p indicates $s_{p'}^{(l)} = 0$. According to Lemma 6, $u_{p'-1}^{(l)} + s_{p'}^{(l)} = c^{(l)} > 0$. So $u_{p'-1}^{(l)} > 0$. Indeed, $u_{p'-1}^{(l)} > u_k^{(l)}$. This justifies that the tree branch can sustain a streaming rate $y^{(l)}$. Step (7) guarantees that the tree spans all users.

The justification of step (11) is similar, and we omit the proof here. □

With Corollary 1 and Lemma 7, we finally can prove Lemma 1.

Proof. By Lemma 7, we have shown that the Bottleneck Removal algorithm can correctly construct a spanning tree which sustains the desired streaming rate as indicated by step (4). It remains to show that the algorithm terminates after a finite number of iterations, e.g., $L \le N$.

To see this, note that by Corollary 1, we know in each iteration, there exists at least one number k such that either $u_k^{(l)}$ becomes zero, or $s_{k+1}^{(l)}$ becomes zero. There are N number of users on the ring, so the total number of iterations cannot exceed N. After at most N iterations, all the residual capacities become zero, and the streaming scheme is realized by a multi-tree consisting of $L \le N$ trees. However, the maximum tree depth can be as large as N. □

10.2 Proof of Lemma 2

Proof. We prove Equation (21). From the definition of $g(k)$ in (17), we have

$$(N-k+1)U_k \le \sum_{i=k+1}^{N} U_i + S + U_k \Rightarrow U_k \le \frac{\sum_{i=k+1}^{N} U_i + S}{N-k}.$$

Then,

$$\begin{aligned} \frac{\sum_{i=k}^{N} U_i + S}{N-k+1} &= \frac{\sum_{i=k+1}^{N} U_i + S + U_k}{N-k+1} \\ &\le \frac{\sum_{i=k+1}^{N} U_i + S + \frac{\sum_{i=k+1}^{N} U_i + S}{N-k}}{N-k+1} = \frac{\sum_{i=k+1}^{N} U_i + S}{N-k}. \end{aligned}$$

Following similar steps, we can prove (22) also. □

10.3 Proof of Lemma 3

Proof. Suppose the Snowball algorithm stops after L_1 steps. We already know that, after the Snowball algorithm stops, we have either (Case 1) all the nodes have been absorbed into the snowball, i.e.,

$$U_1^{(L_1)} = U_2^{(L_1)} = \cdots = U_N^{(L_1)} = u,$$

or (Case 2) all the nodes in the snowball have zero remaining capacity, i.e.,

$$U_{J'+1}^{(L_1)} = U_{J'+1}^{(L_1)} = \cdots = U_N^{(L_1)} = 0 \text{ for some } J' < J.$$

We will consider these two cases separately. Define $H_1 := \lfloor s/r \rfloor$, then we can write

$$s = H_1 r + q, \tag{42}$$

where $q < r$, and we know that $J = N - H_1$.

In Case 1, after step L_1, the stream rate that is already supported is

$$\begin{aligned} \delta^{(L_1)} &= \frac{\sum_{i=1}^{N} U_i - Nu}{J} = \frac{Nr - s - Nu}{J}, \\ &= \frac{Nr - H_1 r - q - Nu}{J} = r - \frac{Nu+q}{J}. \end{aligned}$$

Therefore, the remaining stream rate to support and the residual server load are

$$\begin{aligned} r^{(L_1)} &= \frac{Nu+q}{J}, \\ s^{(L_1)} &= H_1 r + q - H_1 (r - \frac{Nu+q}{J}) = q + \frac{H_1}{J}(Nu+q). \end{aligned}$$

After that, we can construct N substream trees, such that each tree trims J nodes with rate u/J, and each node is trimmed J times. Then after these N substream trees, at the end of step $L_2 = L_1 + N$, we have used up all users' uplink capacity, and the remaining rate to support and the residual server load are

$$\begin{aligned} r^{(L_2)} &= \frac{Nu+q}{J} - \frac{u/J}{N} = \frac{q}{J}, \\ s^{(L_2)} &= q + \frac{H_1}{J}(Nu+q) - NH_1\frac{u}{J} \\ &= q + \frac{H_1}{J} q = N\frac{q}{J} = Nr^{(L_2)}. \end{aligned}$$

Obviously, the stream rate r can be supported in this case, as the server has just enough remaining capacity to upload to all nodes the remaining rate. Therefore, the Snowball algorithm is successful for Case 1.

In Case 2, after the L_1 steps, from (42) and the property $s = \sum_{i=1}^{N}(r-U_i)$, the total supported rate is

$$\begin{aligned}\delta^{(L_1)} &= \frac{\sum_{i=1}^{N} U_i - \sum_{i=1}^{J'} U_i^{(L_1)}}{J} \\ &= \frac{Nr-(H_1 r+q)-\sum_{i=1}^{J'} U_i^{(L_1)}}{J} \\ &= r - \frac{\sum_{i=1}^{J'} U_i^{(L_1)}+q}{J}.\end{aligned}$$

Therefore, the remaining stream rate to support and the residual server load are

$$r^{(L_1)} = r - \sum_{l=1}^{L_1} y^{(l)} = \frac{\sum_{i=1}^{J'} U_i^{(L_1)}+q}{J}, \quad (43)$$

$$s^{(L_1)} = H_1 r + q - H_1 \sum_{l=1}^{L_1} y^{(l)} = q + \frac{\sum_{i=1}^{J'} U_i^{(L_1)}+q}{J} H_1. \quad (44)$$

Note that if the second case occurs, nodes 1 to J' are always trimmed, and thus $r^{(L_1)} \geq U_1^{(L_1)}$.

Now we have the following lemma:

Lemma 8. *Suppose after L_1 steps, the remaining rate to support, the residual peer capacities, and the residual server load are $r^{(L_1)}$, $U_i^{(L_1)}, i = 1,2,\cdots,N$, and $s_{(L_1)}$, respectively. If*

$$U_1^{(L_1)} \leq r^{(L_1)} = \frac{\sum_{i=1}^{J'} U_i^{(L_1)}+q}{J}, \quad (45)$$

then the residual bandwidth (server and peer) can support the remaining rate with a multi-tree algorithm where in each tree, the branch number is at least H_1.

Proof. First, the condition in (45) is necessary, since otherwise, $U_1 < r$ is not fully utilized and the server load cannot be its minimum value.

Second, we show this condition is also sufficient, i.e., whenever (45) is satisfied, there exists an algorithm to support r with s_{min} after the trim-J-whenever-possible strategy stops after L_1 steps. We construct the J' trees in the following way: in the j-th tree, $j = 1,2,\cdots,J'$, nodes i to j are non-leaf nodes and their residual capacities are trimmed, and nodes $j+1$ to N are leaf nodes. The stream rate of the j-th node is $U_j^{(l)} - U_{j+1}^{(l)}$, and as a result, j uses up its capacity after the j-th tree. This way, one substream tree uses up one node's capacity and after the J' trees, the capacities of all nodes are used up, and in this process, the total server load required is

$$\begin{aligned}s' &= U_{J'}^{(l)}(N-J') + (U_{J'-1}^{(l)} - U_{J'}^{(l)})(N-J'+1) \\ &\quad + \cdots + (U_1^{(l)} - U_2^{(l)})(N-1) \\ &= U_1^{(l)}(N-1) - \sum_{i=2}^{J'} U_i^{(l)} = U_1^{(l)} N - \sum_{i=1}^{J'} U_i^{(l)}.\end{aligned}$$

If the condition in (45) is satisfied, then we have

$$\begin{aligned}s' &\leq \frac{N}{J}\left(\sum_{i=1}^{J'} U_i^{(l)} + q\right) - \sum_{i=1}^{J'} U_i^{(l)} \\ &= \frac{N-J}{J}\sum_{i=1}^{J'} U_i^{(l)} + \frac{N}{J} q \\ &= \frac{N-J}{J}\left(\sum_{i=1}^{J'} U_i^{(l)} + q\right) + q = s^{(L_1)},\end{aligned}$$

where the last equality comes from (44). Therefore, the server load required to use up all users' capacities does not exceed the original residual server load. This indicates that our users' uplink loads are maximized, and thus the algorithm is efficient in terms of minimizing server load. It is also easy to see that the stream rate r is supported by the minimum server load s_{min}. □

From Lemma 8, we know that the Snowball algorithm is successful if the second case is reached. Therefore, we have completed the proof of Lemma 3. □

10.4 Proof of Lemma 4

Proof. The proof is straightforward: Denote by s_{min}^h the minimum rate the server needs to upload to nodes in G_h, $h = 1,2,\cdots,H$. Then from Theorem 2, we have

$$s_{min}^h = \max(r, \sum_{i\in B_h}(r-U_i)),$$

and the minimum server load is $s_{min} = \sum_{h=1}^{H} s_{min}^h$.

If (29) is satisfied, then

$$s_{min} = \sum_{i\in B}(r-U_i) = \sum_{i=1}^{N}(r-U_i)^+,$$

and thus the minimum server upload is not affected by the isolated H-division.

In the other direction, if s_{min} is not affected, then $s_{min} = \sum_{i\in B}(r-U_i)$, and we need the condition in (29). □

10.5 Proof of Lemma 5

Proof. First, if $M = 1$, then the nodes form a tandem and the depth is N.

Second, if $M > 1$, then the maximum number of nodes in d levels is $1 + M + \cdots + M^{d-1} = (M^d - 1)/(M-1)$. The minimum depth is the smallest d such that $n \leq (M^d-1)/(M-1)$. Therefore we have $D \geq \lceil log_M((M-1)n+1)\rceil = f(n,M)$.

We next show that $D = f(n,M)$ is achievable. The case for $M = 1$ is trivial, and we only consider the $M > 1$ case here. Consider the following algorithm: $\forall l \in \{1,2,\cdots,n\}$, substream l is such that the server sends only to peer l, and the substream rate is $y^{(l)} = y = \min(U/(n-1), r/n)$. For substream 1, the tree is a depth-first traversal of all peers from 1 to n, as illustrated in Figure 8. For substream tree 2, all the peers perform a left shift, i.e., peer i takes the position of peer $(i-1)_+$ in the first tree, where

$$(i-1)_+ = \begin{cases} i-1 & \text{if } i > 1, \\ n & \text{if } i = 1. \end{cases}$$

For the remaining substream trees, the left-shift strategy continues to apply, i.e., in tree l, each peer i takes the position of peer $(i-1)_+$ in tree $l-1$. So each node takes each position once in the M-ary tree if all the n trees are considered. Therefore the total upload rate of a user in all the substream trees is the same as the total upload

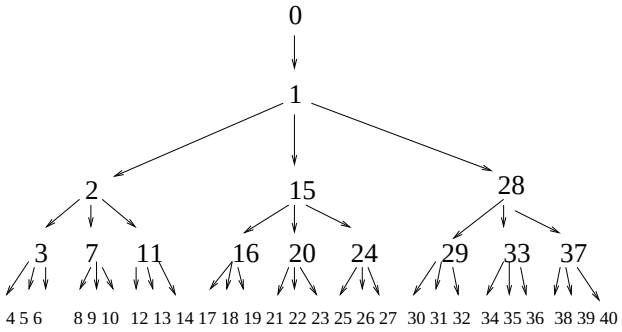

Figure 8: The optimal strategy under M peer constraint. This is an example for $M = 3$, $n = 40$.

rate of all users in a single substream tree. So $u_i = (n-1)y = \min(U,(n-1)r/n)$, for all i.

If $r \leq nU/(n-1)$, then $y = r/n$, and these n substreams are enough to support the streaming rate r. The server's upload rate is $s = nr/n = r = s_{min}$ and for the peers, $u_i = r(n-1)/n \leq U$. If $r > nU/(n-1)$, then for each peer i, $u_i = U$, and the $(n+1)$-th substream of rate $y = r - nU/(n-1)$ is needed, which the server sends to everybody. Then each peer has a download rate

$$d_i = n\frac{U}{n-1} + r - \frac{nU}{n-1} = r,$$

and the server has a upload rate

$$s = n(r - n\frac{U}{n-1}) + n\frac{U}{n-1} = n(r-U).$$

Therefore, a streaming rate of r can be supported with a multi-tree depth of $f(n,M)$. We have $D_{min} = f(n,M)$ in case 2.

Finally, for the depth-first tree and left-shift algorithm, one peer has M children in each substream, and in the superposition of all the N or $N+1$ substreams, one peer has at most $M + (f(n,M) - 2)(M-1)$ children. The reason is that, whenever the node is in depth d, its children are the same, no matter in which substream trees. In the $(f(n,M)-1)$-th level, a node has at most M children. For the above $(f(n,M)-2)$ levels, in each level, one peer has at most $(M-1)$ children not yet considered. Thus combining all the superpositions, one node has at most $M + (f(n,M)-2)(M-1)$ children. □