P2P Streaming Capacity under Node Degree Bound

Shao Liu[1], Minghua Chen [2], Sudipta Sengupta [3], Mung Chiang[1], Jin Li[3], and Phil A. Chou[3]
[1] Princeton University, [2] Chinese University of Hong Kong, [3] Microsoft Research

***Abstract*—Two of the fundamental problems in peer-to-peer (P2P) streaming are as follows: what is the maximum streaming rate that can be sustained for all receivers, and what peering algorithms can achieve close to this maximum? These problems of computing and approaching the P2P streaming capacity are often challenging because of the constraints imposed on overlay topology. In this paper, we focus on the limit of P2P streaming rate under node degree bound, i.e., the number of connections a node can maintain is upper bounded. This is a constraint that is natural in many practical P2P systems but has not been tackled before. We first show that the streaming capacity problem under node degree bound is NP-Complete in general. Then, for the node out-degree bounded case, through the construction of a "Bubble algorithm", we show that the streaming capacity now is at least half of that of a much less restrictive and previously studied case, where we bound the node degree in each streaming tree but not the degree across all trees. Then, for the case with node total degree bounds, we develop a "Cluster-Tree algorithm" that provides a probabilistic guarantee of near-optimality when node degree bound is logarithmic in network size, through a peering construction method that can be implemented in the P2P control plane. The effectiveness of these algorithms in approaching the capacity limit is demonstrated in simulation using uplink bandwidth statistics of Internet hosts and real-world online video viewing traces. Both analysis and numerical experiments show that peering in a locally dense and globally sparse manner achieves near-optimal streaming rate if the degree bound is at least logarithmic in network size.**

I. Introduction

P2P systems have enabled scalable file sharing and video streaming for almost a decade, yet the following fundamental questions on its performance limits have only recently been answered partially: what is the *streaming capacity*, the maximum streaming rate that can be achieved by all receivers of a streaming session, under various overlay topology constraints? What kind of peering algorithms achieve near-capacity rates?

As explained in detail in Section II, P2P streaming can be modeled as multiple multicast trees superimposed on top of the graph of the peers, and the streaming capacity problem depends on the constraints on the graph and tree properties. The simplest case is the unconstrained tree construction on top of a full mesh (complete) graph, where the streaming capacity is derived in several papers [1], [8]–[11]. Various practical constraints on the allowed peering and the given overlay have since been considered. For example, [2], [3] solved the streaming capacity problem under the *node per-tree degree* bound on top of a complete graph. In contrast to *node degree* constraints in this paper, node per-tree degree constraints individually upper bound the number of peers a node can have in each of the multiple trees, thus less related to the practical constraints in system design although easier to tackle mathematically. Then, [6], [7], [12], [13] extended the capacity study to overlay networks that are not full mesh and to those that have "helper nodes" that only act as relays but not receivers. In this paper, we focus on the capacity study under node degree bound.

Node degree is defined as the total number of distinct neighbors across all the multicast trees. Similarly, *node out-degree* is defined as the total number of distinct children of a node. Since a peer is typically a home PC with limited system resources, it is necessary to limit the number of connections (or out-going connections) a node maintains, thus bounding its node degree (or out-degree). Algorithms such as SplitStream and CoopNet [4], [5] bounded node degree but did not provide any streaming rate guarantee. In the theory of P2P streaming capacity, the challenges due to node degree bound, even in the case of full mesh graph without helper nodes, has not been pursued, and becomes the subject of this paper.

Imposing node degree bounds across multiple trees that support the same streaming session makes the combinatorial problem of multi-tree embedding even harder than before. We deploy a combination of graph-theoretic operations and probabilistic arguments to develop two peering algorithms with complementary methodologies and applications:

- First, we show that the streaming capacity problem under node degree bound is NP-Complete in general.
- We prove, in Section III, that the streaming capacity under node out-degree bound (abbrev. as **n.o.d.b.** throughout) is at least half of the capacity under node per-tree degree bound (abbrev. as **p.t.d.b.** throughout). To prove this result, we develop a *Bubble algorithm* that satisfies node degree bound and show that it achieve at least half of the p.t.d.b capacity. Bubble algorithm employs a novel internal-node swap mechanism during construction of trees, bounding the number of distinct children of a node across all the trees. This is the first algorithm that constructs n.o.d.b trees and achieves guaranteed streaming rate.
- We develop in Section IV the *Cluster-Tree algorithm* that can achieve within $1-\epsilon$ of the unconstrained streaming capacity with high probability under large enough node degree bound (abbrev. as **n.d.b.** throughout). The algorithm demonstrates the following insight: peering in a "locally dense, globally sparse" manner is sufficient to achieve near-optimal streaming rate. Performance guarantee of the Cluster-Tree algorithm is derived in part based on the Central Limit Theorem, and only relies on aggregate characteristics of peer uplink bandwidths. So it is robust to individual peer dynamics and mild peer churn. It is the first algorithm that handles peer churns while achieving, in a probabilistic sense, near-optimal streaming rate.

TABLE I
SUMMARY OF RELATED WORK. "W.H.P." IS SHORT OF "WITH HIGH PROBABILITY".

	Graph	*Tree Degree*	*Node Degree*	*Optimality*
Mutualcast [1]	complete, directed	unbounded	unbounded	exact optimality
Snowball [2], [3]	complete, directed	bounded	unbounded	exact optimality
CoopNet / SplitStream [4], [5]	complete, directed	bounded	bounded, arbitrary	no optimality
Iterative in [6]	general, undirected	unbounded	unbounded	$1-\epsilon$ approx.
Iterative in [7]	general, directed	bounded	unbounded	$1-\epsilon$ approx.
Bubble (this paper)	complete, directed	bounded	bounded, arbitrary	$> 1/2$ approx.
Cluster-tree (this paper)	complete, directed	bounded	bounded, $\log N$	$1-\epsilon$ approx. (w.h.p.)

- We demonstrate in Section V the effectiveness of our algorithms through simulations using uplink capacity statistics of Internet hosts, and real-world online video viewing traces. The results show that both algorithms achieve streaming rates that are close to the streaming capacity, for the cases studied in simulations.

Our Cluster-Tree algorithm is a hierarchical scheme. Other hierarchical schemes close to ours include [12] and [13]. In [12], it is proposed to use Mutualcast (where it is called HCPS) both across clusters and within clusters. In [13], the authors use single tree across clusters and Mutualcast within clusters. Our Cluster-Tree algorithm uses degree bounded multiple trees across clusters, and Mutualcast within clusters. It has performance guarantee on throughput and node degree bound; while such guarantee was not provided in [12], [13].

The rest of this paper is organized as follows. After formulating the problem in Section II, we introduce the Bubble algorithm in Section III and the Cluster-Tree algorithm in Section IV. We present our simulation results that validate our analysis in Section V and conclude in Section VI. All proofs are in the Appendix.

II. Modeling and Problem Formulation

The key notation used is summarized in Table II.

TABLE II
KEY NOTATION

Symbol	**Meaning**
s, V, R	the source, set of all nodes, set of all receivers.
C_v	upload capacity of node v in V.
M	per-tree out-degree bound (**p.t.d.b.**).
D_o	node out-degree bound (**n.o.d.b.**).
D	node degree bound (**n.d.b.**).
$\bar{r}_N(C_s, D)$	n.d.b. streaming capacity as a function of C_s and node degree bound D.
$\bar{r}_N(C_s, D_o)$	n.o.d.b. streaming capacity as a function of C_s and node out-degree bound D_o.
$\bar{r}_T(C_s, M)$	p.t.d.b. streaming capacity as a function of C_s and per-tree out-degree bound M.

A. Basic Model of P2P Streaming Systems

There are two classes of P2P streaming approaches, tree-based or mesh-based. Tree-based P2P streaming approaches *explicitly* construct multiple spanning trees (multi-trees) connecting source to all receiver peers, divide the stream into multiple substreams, and route one substream per tree. In mesh-based P2P streaming approaches [12], [13], peers dynamically exchange video packets with several of their neighbors and no explicit tree is constructed. However, even in mesh-based P2P, the trajectory of each packet (or chunk) still forms a spanning tree, originating from the source and reaching each peer exactly once. The difference is really about the lifetime of the multi-trees.

We use a directed graph $G = (V, E)$ to model a P2P network. A vertex v in V is either the source or a receiver peer. An edge $e = (v, u)$ in E represents that peer v is allowed (but not required) to transmit packets to peer u by establishing TCP/UDP connections. If every node is allowed to establish TCP/UDP connections with every other nodes, then G is full mesh [6], [7]. We consider the scenario where a single source s wants to broadcast its data to all receiver peers in $R = V - \{s\}$. Let $N = |R|$ where $|\cdot|$ represents the size of a set.

For each node v in V, its aggregate outgoing rate of is bounded by its uplink capacity C_v, e.g., 384 Kbps for typical ADSL users in the U.S. Similarly, the aggregate incoming rate of node v is bounded by its downlink capacity, e.g., 1.5 Mbps. Similar to most other P2P work [1], [8], [10], [11], [14], we make the "uplink bottleneck" assumption that node uplinks are the only rate limiting bottlenecks in the network. We also assume a peer only stores and forwards its received content.

B. Degree Bounds

Define the degree of a node v as its total number of neighbors, i.e., distinct parents and children across all of the multiple trees. The node degree consists of two parts: *in-degree*, which is the number of distinct parents, and *out-degree*, which is the number of distinct children.

As discussed in Section I, for P2P streaming systems to scale, node degree must be bounded, in order to limit the overhead for a node to maintain concurrent TCP or UDP connections to its neighbors, as well as to bound the fanout delay, the time it takes to forward a packet received from its parent(s) to its children. Here, the fanout delay of node v is defined as the time it takes to forward a packet received from its parent(s) to its children. It is product of the number of children and the transmission time of one packet. Large node out-degree bound D_o allows a node to have large number of children, potentially resulting in severe fanout delay. In recent work, per-tree node degree bounds have been studied. However, the practically more relevant, and algorithmically more challenging, model is to bound the total node degree across all multi-trees, rather than on each tree individually. Indeed, in all practical P2P deployments, the total number of

peers a node can have is bounded. Consider the following two cases:

- We need to bound the *total* node degree bound if TCP is used for communications between peers, since the socket and memory overhead of a node maintaining TCP connections is proportional to the number of both its parents and children.
- On the other hand, we only need to bound node *out-degree* if UDP is used. This is because UDP is connectionless and a receiving node can use one single socket to receive packets from all its parents. Hence, the overhead is independent to the node's in-degree.

Throughout this paper, *we let node degree bound be node total degree bound for TCP and node out-degree bound for UDP*. Similar to [2], [3], we do not cast node degree bound on source s, since it is in general a powerful server with much resource.

Now the question is how much will node degree bounds reduce the P2P streaming capacity, compared to the idealized scenario of no degree bounds and to the less restrictive scenario of per-tree out-degree bound? What is streaming capacity under node degree bound, and can intelligent peering be carried out to approach the limit?

C. Formal Statement of Problem

Consider a full mesh P2P overlay network G and a node degree bound D. Let $G' = (V, E')$ be a graph where the edge set $E' \subseteq E$, and for every node $v \in R$ on G', its node degree is no more than D. G' thus represents a subgraph of G with a node degree bound D. Let $\mathcal{Q}(G, D)$ be the set of all such subgraphs.

Let $\mathbb{T}(G')$ be the set of all feasible spanning trees in G', originating from source s. Let $\mathcal{P}(\mathbb{T}(G'))$ be the power set of $\mathbb{T}(G')$. Every multi-tree $T \in \mathcal{P}(\mathbb{T}(G'))$ is feasible over G'.

Let y_t be the rate of tree t in a chosen multi-tree T. The streaming rate achieved by all receivers is simply $\sum_{t\in T} y_t$. Let the number of children of node v in tree t (its p.t.o.d.) be $m_{v,t}$. Then, The uplink capacity constraint of node v is $\sum_{t\in T} m_{v,t} y_t \leq C_v$, $\forall v \in V$. The question now comes down to a *joint neighbor selection and packing spanning tree problem* as follows:

Problem 1 (Streaming Capacity under Node Degree Bound):

$$\max_{G'\in\mathcal{Q}(G,D), T\in\mathcal{P}(\mathbb{T}(G')), y_t\geq 0, \forall t\in T} \quad \sum_{t\in T} y_t \tag{1}$$

$$\text{s.t.} \quad \sum_{t\in T} m_{v,t} y_t \leq C_v, \quad \forall\ v \in V. \tag{2}$$

The streaming capacity is the optimal value given by solving Problem 1.

The peer upload capacities are fixed. However, the service provider usually has the capability to change upload capacity of the source s. Thus, the streaming capacity under node degree bound is a function of C_s and node degree D, and we denote it by $\bar{r}_N(C_s, D)$.

Computing and achieving the streaming capacity are much easier when there is no degree bound. The streaming capacity for the case of $D = \infty$ is recently characterized by a supply-equal-demand argument as follows [1], [8], [10]:

$$\bar{r}_N(C_s, \infty) = \min\left(C_s, \frac{1}{N}\sum_{v\in V} C_v\right). \tag{3}$$

It can be achieved by packing one depth-1 tree and N depth-2 trees, called *Mutualcast trees* [1] [14] [8], as illustrated in Fig. 1.

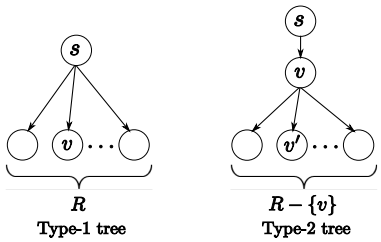


Fig. 1. Type-1 tree has depth 1, and the source s directly broadcast to N recipients. Type-2 trees has depth 2, and s sends one packet to each recipient r, and then r re-broadcasts the packet to the rest $N-1$ recipients.

Since source s is usually a high-end server with large bandwidth, C_s is in general larger than $\frac{1}{N}\sum_{v\in V} C_v$. Let $\mu = \frac{1}{N}\sum_{v\in R} C_v$ be the average receiver upload capacity, then $\bar{r}_N(C_s, \infty)$ approaches μ asymptotically:

$$\bar{r}_N(C_s, \infty) = \frac{1}{N}\sum_{v\in R} C_v + \frac{1}{N} C_s \to \mu \text{ as } N \to \infty. \tag{4}$$

However, when node degree bounds are imposed, it becomes much more challenging to either compute or come close to the streaming capacity. The following proposition shows that Problem 1 is in general hard.

Proposition 1: For general graph G and $D \geq 2$, Problem 1, the Streaming Capacity problem under Node Degree Bound is NP-Complete.

The proof is carried out by showing any instance of the NP-Complete DEGREE-BOUNDED-SUBGRAPH problem [15] can be reduced to the above problem.

In the rest of the paper, we seek polynomial time algorithms that construct node degree bounded trees on top of a full-mesh graph and have performance guarantees. Note that, there are prior works studying the P2P streaming capacity on top of a non-full-mesh overlay graph; see [6] for undirected graph without any degree bound, and [7] for directed graph with p.t.d.b. Those works optimize over tree packing for fixed neighboring relationships and do not bound node degree bound. Compared with those works, our work is *joint optimization over neighbor selection and spanning tree packing.*

Even when G is full mesh (and $D \geq 2$), we conjecture that the problem of Streaming Capacity under Node Degree Bound is still NP-Complete.

III. N.O.D.B Streaming Capacity and Bubble Algorithm

In this section, we study the streaming capacity under node out-degree bound (**n.o.d.b.**). Since this problem is NP-Complete in general, we provide upper and lower bounds of the n.o.d.b. streaming capacity, denoted by $\bar{r}_N(C_s, D_o)$.

The upper bound of $\bar{r}_N(C_s, D_o)$ is straight forward: since n.o.d.b. is a stronger constraint than node per-tree degree bound (**p.t.d.b.**), denoted by $\bar{r}_T(C_s, M)$, we have

$$\bar{r}_N(C_s, D_o) \leq \bar{r}_T(C_s, M)|_{M=D_o}. \quad (5)$$

The node p.t.d.b. streaming capacity $\bar{r}_T(C_s, M)|_{M=D_o}$ is derived in [2], [3], which propose a "Snowball algorithm" that satisfies p.t.d.b. and achieves the exact streaming capacity $\bar{r}_T(C_s, M)$. As the Snowball algorithm achieves $\bar{r}_T(C_s, M)$, we also call it "Snowball capacity". Without loss of generality we assume that $C_1 \geq C_2 \geq \cdots \geq C_N$ for N receiver peers. Then, it is shown in [2], [3] that, $\bar{r}_T(C_s, M) = \bar{r}_N(C_s, \infty)$, i.e., p.t.d.b. does not affect the streaming capacity, if and only if

$$\frac{\sum_{v=2}^{N} C(v)}{N - M - 1} \geq \frac{C(1)}{M}. \quad (6)$$

Otherwise, there exists a C_s such that $\bar{r}_T(C_s, M) < \bar{r}_N(C_s, \infty)$.

For the lower bound of $\bar{r}_N(C_s, D_o)$, we have:

Theorem 1: For a P2P streaming system with N receiver peers and one server with bandwidth C_s, the n.o.d.b. streaming capacity $\bar{r}_N(C_s, D_o)$ is at least half of the p.t.d.b. streaming capacity $\bar{r}_T(C_s, M)$, if $M = D_o$, i.e.,

$$\bar{r}_N(C_s, D_o) \geq \frac{1}{2}\bar{r}_T(C_s, M)|_{M=D_o}, \forall C(s) \geq 0, \forall D_o, N \geq 1. \quad (7)$$

We prove Theorem 1 by constructing a "Bubble algorithm" that satisfies n.o.d.b., and achieves at least half of the Snowball capacity. We introduce some notations before we describe Bubble algorithm, an iterative algorithm: At any stage of the iterations, we use U_v to denote the current total upload rate of node v, and use $\tilde{C}_v = C_v - U_v$ to denote its remaining capacity. We say a node v is larger than a node u if $\tilde{C}_v > \tilde{C}_u$. We say node v is "exhausted" if $\tilde{C}_v = 0$. If v is an internal node in a tree t, then we say tree t is node v's "internal tree".

Bubble algorithm is a greedy algorithm. The main ideas are: 1) We iteratively construct trees with tree degree $M = D_o$, and require that the children set of each node remains unchanged across its internal trees. 2) Once we construct a tree, we assign a maximum allowed rate to this tree until one of the internal node is exhausted, and we swap the exhausted internal node with the largest leaf node and construct new trees. 3) The swap must be in a way such that no new children are brought to an old internal node, otherwise that internal node violates the n.o.d.b. constraint. 4) To maximally utilize peer bandwidths, the server uploads to only one single child if possible, and we call these trees "initial trees". For an initial tree, we call the single child of the server the "root" of this initial tree. We continue the iterations until we do not have enough non-exhausted nodes to construct a M-ary initial tree.

We call it a "Bubble" algorithm, since we can think of the N nodes as N layers of liquids, placed from low to high, with densities equaling to C_1 to C_N initially. When a node is exhausted, we say its corresponding layer of liquid is vaporized to a "bubble". From law of gravity, the light "bubble" node will lift up, and the heaviest liquid above the new formed bubble will fall down, which corresponds to a swapping in our algorithm. We note that Bubble algorithm is a centralized algorithm, and thus is not a practical P2P streaming scheme. We only design it to prove Theorem 1, and it is the first algorithm that satisfies n.o.d.b. and has provable performance guarantee.

A. Bubble Algorithm Description

Bubble algorithm constructs trees with degree $M = D_o$ in an iterative fashion, one tree in each round. Define $I = \lceil \frac{N-1}{M} \rceil$, then any tree with degree M contains at least I internal nodes.

1) We first build a balanced tree, called tree 1, with tree degree M by using the first I nodes as internal nodes; see an example in the first plot of Fig. 4. For the I internal nodes, the first $I-1$ one each has M children and the I-th node has $N - 1 - (I-1)M \in [1, M]$ children. We assign the maximum possible rate in this tree, so that at least one (denote the number by n) of the I internal nodes or the source exhausts.
2) We swap the n exhausted internal nodes of tree 1 with n largest non-exhausted leaf nodes of tree 1, one by one, to form a new tree, named tree 2. We assign the maximum possible rate for tree 2, so that at least one of its internal nodes or the source exhausts.
3) We repeat the swapping process for tree 2 to generate tree 3, and so on.

We terminate the tree construction process if there are less than I non-exhausted nodes, or if the source exhausts.

The key challenge is to swap in a way such that n.o.d.b. is not violated. Suppose node A is an exhausted internal node, and E is the largest original leaf node that will be swapped with A. Consider the relationships between A and E, we have altogether three possibilities, as illustrated in Fig. III-A.

- **Case** 1**:** A is not an ancestor of E; for that case, we can simply give all children of A to E, and keep the other internal nodes' children unchanged.
- **Case** 2**:** A is the parent of E. For that case, we cannot simply swap the positions of A and E, since this way, A's parent will have a new distinct child. Similarly, E cannot be the child of any other internal peer node, otherwise that peer will have a new child. Since the server is not degree bounded, the only choice is to make E the single child of the server, let E take the original root and all the other children of A as its children, and A is left as a leaf node.
- **Case** 3**:** A is an ancestor, but not a parent, of E. For that case, suppose B is the child of A that is an ancestor of E. Similar to the reasoning in case 2, to ensure that no old internal nodes add a new child, we need to make B the new root, and let E takes the original root and all the other children of A as its children.

There are at least $M-1$ trees constructed by Bubble algorithm. This is achieved in the case where in every iteration all the I internal nodes are exhausted simultaneously, and it needs at least $M-1$ iterations to leave less than I nodes unexhausted. The maximum number of Bubble trees is $N - I + 1$, in which case exactly one node is exhausted for each iteration, and it takes $N - I + 1$ trees to leave at most $I - 1$ nodes unexhausted.

If the server capacity is small and the server is exhausted at any stage in these iterations, the tree construction stops, and it is the server bandwidth, not the peer bandwidth, that limit the streaming rate. For this case, the node degree bound does not affect the streaming rate, and the streaming capacity is the same as the unbounded case. If the sever capacity is large enough to survive all the Bubble tree constructions, then after the Bubble algorithm stops and less than I nodes left unexhausted, we have no additional leaf node to swap with an exhausted internal node, and we simple move all the children of the exhausted internal node to the server, and keep the structure of other nodes unchanged.

A complete example of how the Bubble algorithm constructs the multi-tree to utilize peer bandwidths is shown in Fig. 3 and Fig. 4. For this example, $N = 9$, $M = D_o = 3$, and we index the peers from A to J. We set the server capacity to be sufficiently large. The remaining capacities and the nodes being swaped after each iteration are shown in Fig. 3, and the trees constructed in the iterations are shown in Fig. 2.

B. Throughput and Node Degree Analysis

We now study the streaming rate guarantee of Bubble algorithm by comparing the streaming rates of Bubble algorithm and Snowball algorithm [2], [3], denoted by $r_b(C_s, D_o)$ and $r_s(C_s, M)$, respectively. We first have

$$r_s(C_s, M) = \bar{r}_T(C_s, M) \geq \bar{r}_N(C_s, D_o) \geq r_b(C_s, D_o), \text{ if } M = D_o. \tag{8}$$

Therefore, once we lower bound the ratio of Bubble rate over Snowball rate, we both provide the sub-optimality guarantee of Bubble algorithm, and provide a lower bound for n.o.d.b streaming capacity.

Both Snowball and Bubble algorithms use initial trees until not possible, and suppose when initial trees are no longer possible, the Snowball and Bubble algorithms can support streaming rate r_s^* and r_b^*, respectively. We have

$$\begin{aligned} &r_s(C_s, M) = C_s \text{ if } C_s \leq r_s^* \text{ , and } r_s(C_s, M) < C_s \text{ otherwise,} \\ &r_b(C_s, D_o) = C_s \text{ if } C_s \leq r_b^* \text{ , and } r_b(C_s, D_o) < C_s \text{ otherwise,} \end{aligned} \tag{9}$$

and we call r^* the critical rate for the two algorithms, and we have the following lemma:

Lemma 1: If $D_o = M$, then we have the following comparisons of Snowball and Bubble algorithms:

$$r_s^* \geq r_b^* \tag{10}$$

$$\frac{\bar{r}_N(C_s, D_o)}{\bar{r}_T(C_s, M)} \geq \frac{r_b(C_s, D_o)}{r_s(C_s, M)} \geq \frac{r_b(r_s^*, D_o)}{r_s(r_s^*, M)} \geq \frac{r_b(r_b^*, D_o)}{r_s(r_s^*, M)} = \frac{r_b^*}{r_s^*} \tag{11}$$

Inequality (10) and the first inequality in (11) are straightforward from (8), the third inequality in (11) is straightforward from (10), and the last equality in (11) is straightforward from (9). The key result is the second inequality in (11), which concludes that the ratio of Bubble rate over Snowball rate reaches its smallest value when $C(s) = r_s^*$. This feature is illustrated in Fig. 5. From Lemma 1, once we bound r_s^*/r_b^*, we bound the ratio between Bubble rate and Snowball rate, and thus bound the ratio between node degree bounded streaming capacity and p.t.d.b. streaming capacity.

Proposition 2: Consider a single-source P2P streaming scenario with N receivers and n.o.d.b. $D_o = M$, and define $I = \lceil (N-1)/M \rceil$. We have

$$\frac{r_b^*}{r_s^*} \geq \frac{I}{2I-1}, \tag{12}$$

$$\frac{r_b(C_s, D_o)}{r_s(C_s, M)} \geq \frac{I}{2I-1} + \frac{I-1}{(2I-1)(1+M)}. \tag{13}$$

The inequality in (12) concludes that the ratio of the two critical rates is at least $1/2$, and (13) comes from (12).

From Lemma 1 and Proposition 2, we have:

$$\bar{r}_N(C_s, D_o) \geq r_b(C_s, D_o) \geq \frac{1}{2}\bar{r}_T(C_s, M)|_{M=D_o} \tag{14}$$

$$r_b(C_s, D_o) \geq \frac{1}{2}\bar{r}_T(C_s, M)|_{M=D_o} \geq \frac{1}{2}\bar{r}_N(C_s, D_o) \tag{15}$$

Equation (14) states that the n.o.d.b. streaming capacity is at least half of the p.t.d.b. streaming capacity, which is Theorem 1. Equation (15) states that Bubble algorithm guarantees a $1/2$ sub-optimality in term of rate performance.

Proposition 2 gives the performance guarantee of Bubble algorithm for the worst case. The critical rates and streaming rate curves for Bubble, Snowball (capacity under per-tree out-degree bound) and Mutualcast (capacity under no bound at all) algorithms are illustrated in Fig. 5.

The worst case occurs if there are N peers, $2I-1$ of them have the same positive bandwidth C, and the others have zero bandwidth. For this worst case, the first Bubble tree exhausts the first I nodes, while there is not enough unexhausted nodes to construct the second Bubble tree, and $r_b^* = IC/(N-1)$. For this worse case, Snowball algorithm can achieve a rate that is the same as unbounded capacity $\bar{r}_N(C_s, \infty)$, and $r_s^* = (2I-1)C/(N-1)$. Even with n.o.d.b., we can easily construct another tree that has all the $2I-1$ nodes be internal with the same out-degree (the last node could possibly be an exception), and this tree can almost achieve the unbounded capacity $\bar{r}_N(C_s, \infty)$. Therefore, for the worst case, $r_b^*/r_s^* \to 1/2$ and $r_b(C_s, D_o)/\bar{r}_N(C_s, D_o) \to 1/2$ as both N and M approaches to infinity.

Although $1/2$ is the worst case ratio, we will show below that the ratio of $r_b(C_s, D_o)/\bar{r}(C_s, \infty)$ is actually very close to 1 for most realistic peer bandwidth distributions.

C. Special Case: Homogeneous Receivers

We study a special case where all receiver nodes have the same upload capacities. Results for this case are not only of independent interest, but also useful later in Section IV. In general, Bubble trees are not interior-node-disjoint. However, for the case of homogeneous receivers, it is true that interior nodes of these Bubble trees are disjoint. This is because on one Bubble tree, all interior nodes use up their capacities simultaneously, and are leaf nodes in all the rest trees. We summarize this and other results in the following proposition.

Proposition 3: Consider constructing degree-M Bubble trees in a single-source streaming scenario with N homogeneous receivers, i.e. $C_v = C$ for all v in R, and $N > M+1$. The followings are true:

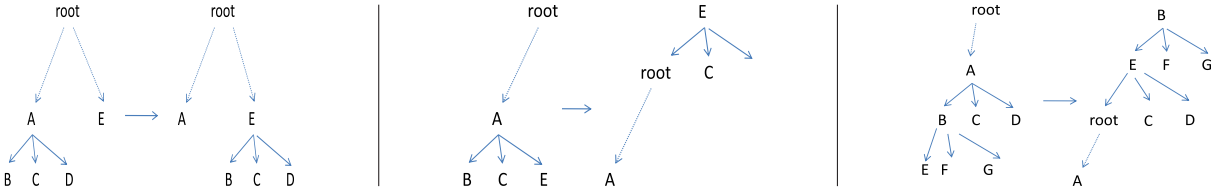


Fig. 2. The swap of an exhausted internal node and a leaf node. From left to right, case 1, case 2 and case 3. Case 1: A is not an ancestor of E. Case 2: A is the parent of E. Case 3: A is an ancestor, but not the parent, of E.

Example 1: $N = 10, C = [20, 15, 10, 8, 7, 6, 5, 4, 3, 2], M = 3$.

$\tilde{C}(A)$	$\tilde{C}(B)$	$\tilde{C}(C)$	$\tilde{C}(D)$	$\tilde{C}(E)$	$\tilde{C}(F)$	$\tilde{C}(G)$	$\tilde{C}(H)$	$\tilde{C}(I)$	$\tilde{C}(J)$	y_t	aggregate rate	$d_{s,o}$	U_s
20	15	10	8	7	6	5	4	3	1.5	10/3	10/3	1	10/3
10	5	(0)	(8)	7	6	5	4	3	1.5	5/3	15/3	1	15/3
5	(0)	0	3	(7)	6	5	4	3	1.5	3/3	18/3	1	18/3
2	0	0	(0)	4	(6)	5	4	3	1.5	2/3	20/3	1	20/3
(0)	0	0	0	2	4	(5)	4	3	1.5	2/3	22/3	1	22/3
0	0	0	0	(0)	2	3	(4)	3	1.5	2/3	24/3	1	24/3
0	0	0	0	0	(0)	1	2	(3)	1.5	1/3	25/3	1	25/3
0	0	0	0	0	0	(0)	1	2	(1.5)	1/3	26/3	1	26/3
0	0	0	0	0	0	0	0	1	0.5	1/6	26.5/3	4	28/3
0	0	0	0	0	0	0	0	0.5	0	1/6	27/3	7	31.5/3
0	0	0	0	0	0	0	0	0	0				

Fig. 3. One example showing the Bubble algorithm. In the table, $\tilde{C}(A)$ to $\tilde{C}(J)$ give the remaining capacities of A to J before each iteration, $d_{s,o}$ represents the source out-degree in the tree constructed in each iteration, and U_s represents the total server load until the end of each iteration. The number with underline means this node is internal. The $(\cdot)$ means the node is to be swaped ((0) means the node is the new formed bubble from the previous iteration).

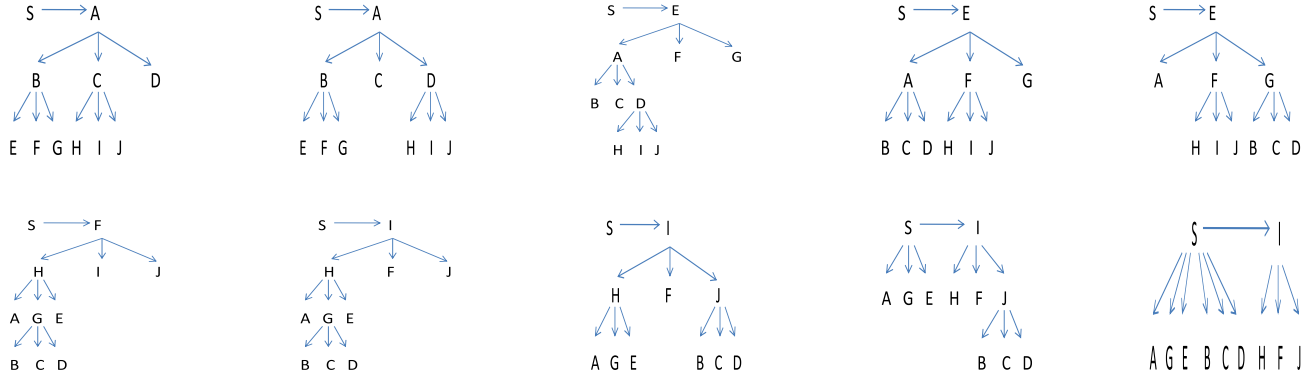


Fig. 4. Constructed trees in all iterations for the example in Fig. 3. The first 8 trees are initial trees. After iteration 8, there are less than $I = 3$ peers left unexhausted, and Bubble algorithms stops. Then server takes care of the children of all exhausted internal nodes from then on.

1) Total number of Bubble trees is either $M-1$ or M, and the trees are interior-node-disjoint;
2) Node degree is bounded by $2M$;
3) Streaming rate supported by these trees is at least $(1 - 1/M)C$.

It is not difficult to see that, in this homogeneous receivers case, the Bubble trees we build are the same as those built by SplitStream [4] and CoopNet [5]. As such, Bubble trees share the same properties as SplitStream and CoopNet trees, such as robustness to peer churn and low delay delivery. We note that CoopNet and SplitStream are practical algorithms with n.o.d.b., but they do not provide any performance guarantee: the streaming rate achieved by them could be far away from the streaming capacity when the peer uplink bandwidths are very different. These advantages together with the guaranteed performance in Proposition 3 make Bubble tree attractive in homogeneous receivers case.

IV. Cluster-Tree Algorithm

In Section III, we have introduced the Bubble algorithm, which provides a *deterministic* guarantee of the streaming capacity under n.o.d.b. However, Bubble algorithm does not bound the node (total) degree bound (n.d.b.), and it is a centralized algorithm. In this section, we study the streaming capacity problem under node (total) degree bound (**n.d.b.**), and provide a *probabilistic* performance guarantee by introducing a Cluster-Tree algorithm, which is a practical design: it is a distributed algorithm and it can handle peer churns.

Assume that C_v ($v \in R$) are i.i.d. with finite mean μ and variance σ^2. Our proposed Cluster-Tree algorithm is a two-level hierarchical scheme that can support a streaming rate of $(1-\epsilon)\mu$ for any ϵ in $(0, 1]$ with high probability, at the same time the node degree is maintained at $D = O(\ln N)$. According to (4), we have for large N,

$$(1-\epsilon)\mu \approx (1-\epsilon)\bar{r}_N(C_s, \infty) \geq (1-\epsilon)\bar{r}_N(C_s, D)\,.$$

Hence, the supported streaming rate of the Cluster-Tree algorithm is close-to-optimal with high probability for large scale P2P systems.

The basic idea is to group peers into clusters, form a full-mesh to deliver content locally within a cluster, and form degree bounded trees to deliver content globally across clusters. Locally, the full-mesh supports high streaming rate within each cluster. The rates supported by different clusters are roughly the same if clusters are large enough, according to the Central Limit Theorem. Globally, using degree bounded Bubble trees (essentially CoopNet/SplitStream trees) to deliver content across these equal-rate clusters achieves high throughput, according to Proposition 3. An illustrative example is shown in Fig. 6.

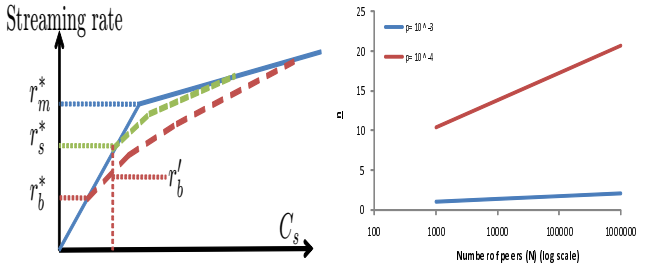


Fig. 5. Plots explaining Theorem 1 and Theorem 2. Left: The critical rates and streaming capacity functions of Bubble (bottom red curve), Snowball (middle green curve) and Mutualcast (top blue curve) algorithms. r^*_m, r^*_s, r^*_b are the corresponding critical rates. r'_b is the streaming rate of Bubble algorithm when $C_s = r^*_s$. The worst $r_b(C_s, D_o)/r_s(C_s, M)|_{M=D_o}$ ratio occurs at $C_s = r^*_s$, and it is r'_b/r^*_b. Right: The curve of $\underline{n}$ versus N for fixed $p = 10^{-3}$ or 10^{-4}. For each p value, we set parameters such that $\frac{2(1+\alpha)\sigma^2}{\epsilon^2\mu^2}$ is the smallest value for (19) to be satisfied for all N from 1000 to 1000,000. It shows that a cluster size of below 25 is enough to guarantee a high probability for system size up to one million.

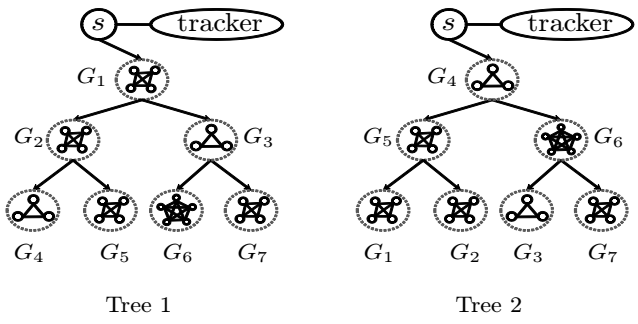


Fig. 6. In this example, the tracker groups the peers into 7 clusters. The server s forms two interior-node-disjoint trees, each having tree degree 2, to distribute video to all clusters. Within each cluster, the peers form a full-mesh and locally broadcast the video among themselves. In tree 1, G_1, G_2, G_3 are interior clusters; while in tree 2, they become leaf clusters and G_4, G_5, G_6 become the interior clusters. There are not enough clusters left to build a third tree with disjoint interior clusters; hence, some clusters, e.g. G_7, do not get the chance to serve as interior clusters.

A. Cluster-Tree Algorithm Description

Trackers have been adopted in many practical P2P streaming systems, such as PPLive [16] and PPStream [17], to provide peer registration and neighbor lookup service.

In these systems, when a node joins the P2P system, it registers itself with the tracker and retrieves a neighbor list from the tracker. The peer node then communicates with the peers in the list to obtain additional lists, and merges these lists to build its own. It connects to both the server and a subset of the neighbors to start watching the video. When a node leaves the system, the tracker is either notified by the node or finds out by periodic inquiry. In these practical systems, the heavy load of the centralized tracker is taken care of by using high-end servers. According to [18], a small number of tracker servers are able to manage possibly millions of streaming users.

In the Cluster-Tree scheme, the tracker is also responsible for 1) grouping nodes into clusters, 2) keeping track of the number of peers in each cluster, 3) appointing cluster heads for each cluster, and 4) building cluster-level Bubble trees.

1) Cluster: Given that N peers want to receive content from source s, the tracker evenly distributes them into clusters. Let K be the total number of clusters, $G_1, \ldots, G_K$ be the clusters, and $n_i = |G_i|$. Define X_i as the average peer upload capacity in G_i, i.e. $X_i = \sum_{v \in G_i} C_v/n_i$. Since C_v are i.i.d. random variables, X_i follows Gaussian distribution with mean μ and variance $\frac{\sigma^2}{n_i}$ when n_i is large. Let $\underline{n} = \min_{1 \le i \le K} n_i$ and $\underline{X} = \min_{1 \le i \le K} X_i$.

Within each cluster, the peers form a full-mesh. If a peer receives data from outside the cluster, it constructs Mutualcast trees to locally broadcast the data to the rest of the peers in the cluster. Consequently, the maximum streaming rate that can be supported within cluster i is X_i. We summarize a useful observation in the following proposition.

Proposition 4: If cluster G_i gets content from outside at rate X_i, then not only it can locally broadcast the content to all peers inside G_i, but it also can deliver the entire or partial content to other clusters at rate X_i.

The tracker appoints cluster head nodes in each cluster. The head nodes are responsible for receiving content from outside, locally broadcasting it to peers in the same cluster, and serving content to outside. The aggregate upload capacity of the head nodes in cluster G_i must be no less than $2X_i$. This is to guarantee that they can broadcast in the cluster at rate X_i, and at the same time send out content to outside at rate X_i. The following proposition shows that at most two head nodes per cluster suffice.

Proposition 5: For any cluster G_i with at least two nodes, either there exists a node $v_1 \in G_i$ so that $C_{v_1} \ge 2X_i$, or there exists two nodes $v_1, v_2 \in G_i$ such that $C_{v_1} + C_{v_2} \ge 2X_i$.
We illustrate the cases of having one cluster head and two cluster heads in Fig. 7.

2) Bounded Degree Trees: Treating the K clusters as *homogeneous* clusters *with all their upload capacities being* $\underline{X}$ (then a cluster is a virtual node in the top hierarchy), the tracker forms multiple degree-bounded M-ary Bubble trees to distribute content to these K virtual nodes (homogeneous clusters). Let M be the n.o.d.b the tracker uses to construct the Bubble trees.

For $M = 1$, the tracker simply arranges the server and the K clusters in a chain, like a bucket brigade. The streaming rate of $\underline{X}$ can be supported and the degree adds to every cluster head is at most 2.

The analysis for the case of $M = 1$ is straightforward. Consider $M \ge 2$. The tracker forms interior-cluster-disjoint

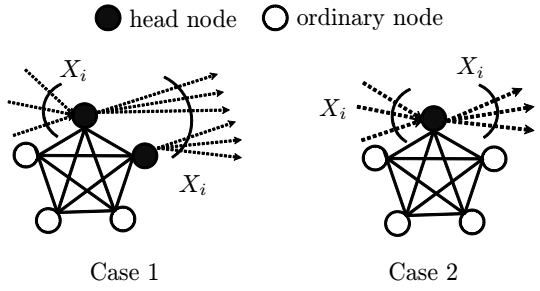


Fig. 7. At most two head nodes per cluster suffice. In case 1, two nodes with aggregate upload capacity more than $2X_i$ server as the cluster head. In case 2, one node with upload capacity more than $2X_i$ servers as the cluster head. Note the receiving head needs to broadcast the received content at rate to peers in the cluster, and this consumes X_i of its upload capacity.

Bubble trees across K homogeneous clusters. As discussed in Section III-C, every cluster serves as interior cluster only once. According to Proposition 3, for $K \geq M+1$, which is almost always true in practice, the total number of trees constructed is bounded by M, and thus the inter-cluster degree of a cluster head can be shown to be bounded by $3M$.[1]

The server splits its video into multiple substreams, and distributes one substream along one Bubble tree. Each interior cluster in a Bubble tree receives a substream at rate $\frac{1}{M}\underline{X}$, and replicates M copies to its M child clusters. This interior cluster becomes leaf cluster in all other trees, and receives distinct substreams at rate $\frac{1}{M}\underline{X}$ per tree. Since there are at least $M-1$ trees, every cluster receives at a total rate of at least $\frac{M-1}{M}\underline{X}$.

For either cases of $M=1$ or $M \geq 2$, the streaming rate of at least $\frac{M-1}{M}\underline{X}$ can be supported,

B. Throughput Analysis

The minimum rate supported by Cluster-Tree scheme with $M \geq 2$, defined as $r_{CT}(M)$, is given by

$$r_{CT}(M) = \frac{M-1}{M}\underline{X}. \tag{16}$$

Since $\underline{X}$ is bounded by μ, we have $r_{CT}(M)$ is upper bounded by μ if $M=1$ and by $\frac{M-1}{M}\mu$ if $M \geq 2$. For $\epsilon \in (0,1]$, we define outage as the event that the $r_{CT}(M)$ for a particular cluster configuration is $\epsilon \cdot 100\%$ percentage away from $\frac{M-1}{M}\mu$. The outage probability, denoted by $P(\epsilon, M)$, is given by

$$P(\epsilon, M) = P\left(\bar{r}_{CT}(M) < (1-\epsilon)\frac{M-1}{M}\mu\right) = P(\underline{X} < (1-\epsilon)\mu). \tag{17}$$

The following theorem shows $r_{CT}(M)$ can be close to μ with high probability, by constructing large enough clusters. For numerical illustration, the right plot of Fig. 5 shows the $\underline{n}$ versus N relationship in (18) with parameters that satisfy the condition in (19).

Theorem 2: If N peers, with independently identically distributed upload capacities, are evenly distributed into $K = \lceil N/\underline{n} \rceil$ clusters where

$$\underline{n} = \left\lceil \frac{2(1+\alpha)\sigma^2}{\epsilon^2\mu^2} \ln N \right\rceil \tag{18}$$

for some constants $\alpha > 0$ and $\epsilon \in (0,1]$, then

$$P(\epsilon, M) = P(\min_{1\leq i\leq K} X_i < (1-\epsilon)\mu) < \frac{1}{2+2\alpha}\left(\frac{\epsilon\mu}{\sigma}\right)^2 \frac{1}{N^\alpha \ln N}, \tag{19}$$

which diminishes to zero as N goes to infinity.

Intuitive explanation: On the one hand, $\underline{n}$ (note $\underline{n} \leq n_i \leq \underline{n}+1$) should be small to keep the node degree small. On the other hand, each cluster should be large enough, so that the Central Limit Theorem applies and $\underline{X}$ is large. It turns out that $P(\underline{X} < (1-\epsilon)\mu)$ is proportional to $e^{-\underline{n}}$. Therefore, it is sufficient to maintain $\underline{n}$ on the order of $\ln N^{(1+\alpha)}$ to make $P(\underline{X} < (1-\epsilon)\mu)$ diminishes to zero as N goes to infinity.

Remarks: 1) Let $M = O(\ln N)$, then the minimum supported rate $r_{CT}(M) \geq (1-\epsilon)\mu$ with high probability for large N. 2) As long as every X_i sufficiently concentrates around μ, then $\underline{X}$ stays close to μ. This indicates throughput of the Cluster-Tree scheme is robust to the independent upload capacities assumption. 3) Throughput of the Cluster-Tree scheme only depends on X_i, which is the average upload capacity of peers in the cluster. As such the throughput of the Cluster-Tree scheme is robust to the peer upload capacity variation, which could be caused by cross traffic introduced by other applications running on the same peer.

The Cluster-Tree algorithm can support a streaming rate of $(1-\epsilon)\frac{M-1}{M}\mu$ with high probability, by allocating N peers into K clusters so that X_i is close to μ for all $1 \leq i \leq K$.

A natural question the becomes: can we achieve a rate of $\frac{M-1}{M}\mu$ with probability one, by forming the K clusters in a way such that $X_i = \mu, \forall 1 \leq i \leq K$. Unfortunately, it is *NP-Complete* to achieve this goal. In particular, determining achievability of the goal is equivalent to solving an average-bin-weight problem, which is NP-Complete according to the following.

Lemma 2: Suppose we are given $K \geq 2$ bins and N balls each having nonnegative weight $w(i), 1 \leq i \leq N$. Consider the following average-bin-weight problem of determining the feasibility of distributing N balls into K bins so that

- each bin contains at least one ball,
- and the average ball weights in all bins are the same.

The average-bin-weight problem is **NP-Complete**.

C. Node Degree and Delay Bound

The total node degree consists of two parts: intra-cluster degree and inter-cluster degree. From Section IV-A, intra-cluster degree is bounded by cluster size $\underline{n}+1$, and inter-cluster degree is bounded by $3M$. Overall, the total node degree is bounded by

$$\underline{n} + 1 + 3M = \left\lceil \frac{2(1+\alpha)\sigma^2}{\epsilon^2\mu^2} \ln N \right\rceil + 1 + 3M. \tag{20}$$

As an example, consider the case where $\epsilon = \frac{3}{7}$, $\sigma = \frac{1}{2}\mu$, $\alpha = \frac{1}{2}$, $M = 8$ and $N = 10^6$. The node degree bound computed by (20) is 87. The probability that the target streaming rate

[1]For any cluster head, its out-degree is bounded by M. Its in-degree is bounded by $2M$, since there are at most M Bubble trees, and one virtual node (cluster) has at most two head nodes. Therefore, the overall inter-cluster degree a cluster head maintains across multiple trees is bounded by $3M$.

0.5μ cannot be supported by the system is 1.5×10^{-5}, according to (19) in Theorem 2. Since the nodes are evenly distributed into clusters, every cluster has size of at least 70 computed by (18) and at most 71. The total number of clusters is 14285.

We next study bound on delay. There are a few flavors of delay in P2P streaming, and there have been several works studying the bound of delay [11], [19], [20]. The delay we focus is the playback delay, which is defined as the time gap from a packet originating at source to it reaching the last receiver. This delay is a function of both propagation delay and *fanout* delay at nodes, as well as the size of the packet being distributed. We assume propagation delays of all overlay links are the same, denoted by z_d. It is shown in [7] that fanout delay of a node v with M children on a tree is given by ML/C_v where L is the packet length, and the node's fanout delays across multiple trees do not interfere with each other.

In the Cluster-Tree scheme, usually K is far larger than M. The maximum number of hops a packet travels across clusters is the depth of cross-cluster trees, i.e. $\log_M K$ for $M \geq 2$. For each hop, the packet experiences a fanout delay of at most $ML/\min_{v\in V} C_v$. Within a cluster, a packet travels at most two hops before reaching all local peers. For each hop, the fanout delay the packet experience is bounded by $(\underline{n}+1)L/\min_{v\in V} C_v$.

Therefore, the overall playback delay of the Cluster-Tree scheme with $M \geq 2$ is upper bounded by

$$\begin{cases} \left(3z_d + (2\underline{n}+3)\frac{L}{\min_{v\in V} C_v}\right)\lceil N/\underline{n}\rceil, & \text{if } M = 1; \\ \left(3z_d + (2\underline{n}+2+M)\frac{L}{\min_{v\in V} C_v}\right)\log_M\lceil N/\underline{n}\rceil, & M \geq 2. \end{cases} \tag{21}$$

Here, $3Z_d$ is because the propagation from one cluster to another is at most 3 hops: at most 2 hops within one cluster, and 1 hop between the two clusters. Note that the bound in (21) is very conservative. It assumes that, the total delay from any cluster to any other cluster is the worst case delay. In reality, the end-to-end delay could be much smaller than the bound in (21), especially if one head node is enough for many clusters. Combining the above playback delay bound and Theorem 2, we observe that varying $\underline{n}$ can result in trading streaming rate with playback delay.

Given N, M and α, large $\underline{n}$ leads to small ϵ according to (18) and hence high $(1-\epsilon)\mu$. However, according to (21), large $\underline{n}$ will also lead to large playback delay. It is clearly a trade-off between streaming rate and playback delay, as controlled in part by the cluster sizes and consequently the node degrees. For example, [20] derived a bound on minimum playback delay that is smaller than the bound in (21). However, the streaming rate achievable under the minimum playback delay above is significantly lower than the streaming capacity achievable through the Cluster-Tree algorithm.

We summarize the impact of different parameters on the throughput, node degree, and delay in Table IV-C.

Parameter	Lower bound of $r_{CT}(M)$	Upper bound of $P(\epsilon, M)$	Node degree	Delay
ϵ	$\downarrow$	$\uparrow$	$\downarrow\downarrow$	$\uparrow$
M	$\uparrow$	$\downarrow$	$\uparrow$	$\downarrow\downarrow$
α	N/A	$\downarrow\downarrow$	$\uparrow$	$\downarrow$

TABLE III
IMPACT OF ϵ, M, AND α ON THE PERFORMANCE MEASURES OF THE CLUSTER-TREE SCHEME. THE SYMBOL $\uparrow$ MEANS THE PERFORMANCE MEASURE INCREASES AT MOST LINEARLY IN THE PARAMETER. THE SYMBOL $\downarrow$ MEANS THE PERFORMANCE MEASURE DECREASES AT MOST LINEARLY IN THE INVERSE OF THE PARAMETER. THE SYMBOL $\uparrow\uparrow$ MEANS THE PERFORMANCE MEASURE INCREASES AT LEAST QUADRATICALLY IN THE PARAMETER. THE SYMBOL $\downarrow\downarrow$ MEANS THE PERFORMANCE MEASURE DECREASES AT LEAST QUADRATICALLY IN THE INVERSE OF THE PARAMETER.

D. Bootstrap and Peer Churn

In previous description and analysis, we assume the swarm size N is large and show the Cluster-Tree scheme can achieve good throughput, low node degree, and small delay. In this section, we study how the Cluster-Tree scheme bootstraps, i.e. how the system gradually evolves, while maintaining good performance, as N grows from zero. We also discuss how the system adjusts itself in the presence of peer churn.

Initially, the tracker assumes an upper bound for the number of nodes in the system, denoted by $\bar{N}$, e.g., 10^6. It gets μ and σ^2 from a third-party statistics [21], and picks the streaming rate $(1-\epsilon)\mu$ and a desired outage probability by assuming $\bar{N}$ nodes in the system. It picks Bubble tree degree M to warranty a low playback delay, and reverse-engineers the required α by using (19). Finally, the tracker computes the minimum required cluster size $\underline{n}$ for a system with $\bar{N}$ nodes according to (18). According to the discussion right after Theorem 2, forming clusters with size n^* guarantees that the desired throughput is achieved with high probability whenever $N \leq \bar{N}$.

As nodes start to join, the tracker groups them into a single cluster. As the number of nodes in the cluster grows beyond a critical size $2n^*$, the tracker splits the cluster into two, each with size n^*. As more nodes join, they get added into existing clusters, and clusters grow and split whenever critical size $2n^*$ is reached. This way, the system bootstraps and at the same time maintains the desired streaming rate with high probability. The largest degree of a node during the process is at most $2n^* + 1 + 3M$.

With peer churn, the cluster size may decrease below $\underline{n}$. The tracker should make sure the minimum cluster size remains larger than n^*. Whenever a peer joins, the tracker assigns it to the minimum-size cluster at that time. If two clusters both have small sizes, the tracker can merge them to a larger size cluster. When a peer leaves, some cluster may shrink below n^*. In this case, the tracker merges two small size clusters to get a new larger size cluster.

For the join and leave of peers, and the merging and splitting of clusters, all are equivalent to adding or deleting Mutualcast trees within a cluster, which are manageable operations. Besides forming clusters, the tracker also needs to maintain M-ary Bubble (essentially CoopNet) trees across clusters as the number of clusters varies due to peer churn. The source and at most two head nodes per cluster are involved. The associated overhead is the same as CoopNet/SplitStream trees [4], [5]. since our cross-cluster Bubble trees are the same as SplitStream and CoopNet trees and thus can be maintained using the same set of adjustment actions.

The pseudo code of the Cluster-Tree algorithm is given in Alg. 1.

Algorithm 1 : Cluster-Tree algorithm.

// given critical cluster size n^*
// initial values: $K = 1$, $N = 0$
if a new peer v joins **then**
  $i_{min} = \text{argmin}_{1 \le i \le K} n_i$;
  $G_{i_{min}} = G_{i_{min}} \cup \{v\}$; $n_{i_{min}} = n_{i_{min}} + 1$;
  adjust local Mutualcast trees within $G_{i_{min}}$;
  if $n_{i_{min}} \ge 2n^*$ **then**
    split $G_{i_{min}}$ into two clusters with equal size n^*;
    assign top two largest capacity peers in each cluster as the cluster heads;
    build local Mutualcast trees in the new clusters;
    adjust cluster-level Bubble trees;
    $K = K + 1; N = N + 1$;
  end if
else if a peer v leaves from G_j **then**
  $G_j = G_j - \{v\}$; $n_j = n_j - 1$;
  if peer v is a cluster head **then**
    assign top two largest capacity peers in G_j as the cluster heads;
  end if
  rebuild/adjust local Mutualcast trees within G_j;
  if $n_j < n^*$ **then**
    $i_{max} = \text{argmax}_{1 \le i \le K} n_i$;
    merge G_j and $G_{i_{max}}$;
    assign top two largest capacity peers in each cluster as the cluster heads;
    build local Mutualcast trees in the new cluster;
    adjust cluster-level Bubble trees;
    $K = K - 1; N = N - 1$;
  end if
end if

V. Evaluation on P2P Topologies

The algorithms developed in this paper can be embodied into the control/management plane of P2P streaming systems. In our numerical simulations, we consider networks with large number of nodes and choose the node uplink capacities randomly from a distribution that is obtained from real peer usage data as reported in [21]. The possible uplink capacities of peers and their respective fractions in the peer population is summarized in Table V. The behavior for different values of N, ranging from 10 to 1,000,000, is similar, hence we present results for $N = 100,000$, unless otherwise stated.

Uplink Capacity (Kbps)	64	128	256	384	768
Fraction (%)	2.8	14.3	4.3	23.3	55.3

TABLE IV
Peer Uplink Capacity Distribution

A. Bubble Algorithm

We first compare the streaming rate r versus service capacity C_s curves for four different algorithms: Bubble (algorithm developed in Sec III of this paper), Snowball [2], [3] (achieving per-tree out-degree bound streaming capacity), CoopNet/SplitStream [4], [5] (node degree bounded, but no performance guarantee), and Mutualcast [1] (achieving unbounded streaming capacity), and they are demonstrated in the left plot of Fig. 8, which shows that the Bubble rate curve is close to that of Mutualcast, or the non-bounded streaming capacity, if $D_o = M = 10$ (throughout the simulations, we use M to denote node degree bound also). We next demonstrate the impact of M on the performance of Bubble algorithm in the right plot of Fig. 8 (varying $M = 1, 2, 4, 6$). From the plot, we see that larger M leads to better streaming rate, which is intuitive. Furthermore, the improvement becomes marginal as $M > 4$ (beyond $M = 6$, the different curves almost coincide, hence not shown), which means that as long as M is not too small, Bubble algorithm gives very good performance for practical peer uplink bandwidth distributions.

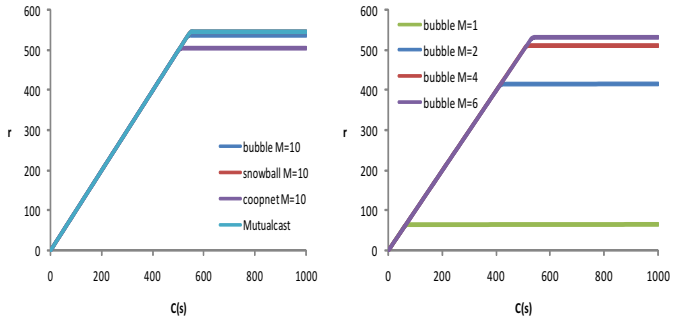


Fig. 8. Left: Streaming rate r vs. source uplink $C(s)$ for different algorithms, $N = 100,000$, $M = 10$. Right: Streaming rate vs. source uplink $C(s)$ for Bubble algorithm, $N = 100,000$, $M = 1, 2, 4, 6$.

The next set of graphs evaluate the ratios achieved by the Bubble algorithm with respect to Mutualcast algorithm, which gives the degree unbounded streaming capacity. We plot the rates achieved by Bubble, Snowball, and CoopNet algorithms as a fraction of that achieved by Mutualcast. Left plot in Fig. 9 shows that for $M = 2$, Bubble performs fairly close to Coopnet and is within 25% of the rate achieved by Snowball or Mutualcast, hence within 25% of optimal. For $M = 6, 10$, Bubble is within 3% of optimal and does better than CoopNet by about 5-10%, as seen in center and right plots of Fig. 9. The important observation is that on practical peer-to-peer topologies like the one used in these simulations, Bubble performs significantly better than the established worst case bound of 50% optimal.

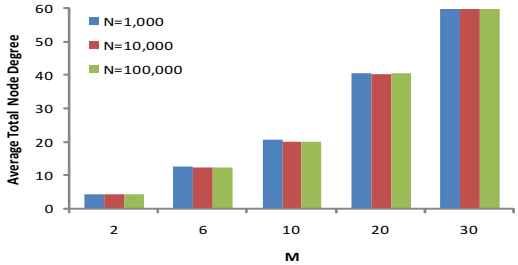


Fig. 10. Average total node degree in Bubble algorithm for $N = 1000, 10000, 100000$ nodes, degree bound $M = 2, 6, 10, 20, 30$.

In our theoretical analysis, we did not bound the total degree (indegree and outdegree) of a peer node in the Bubble

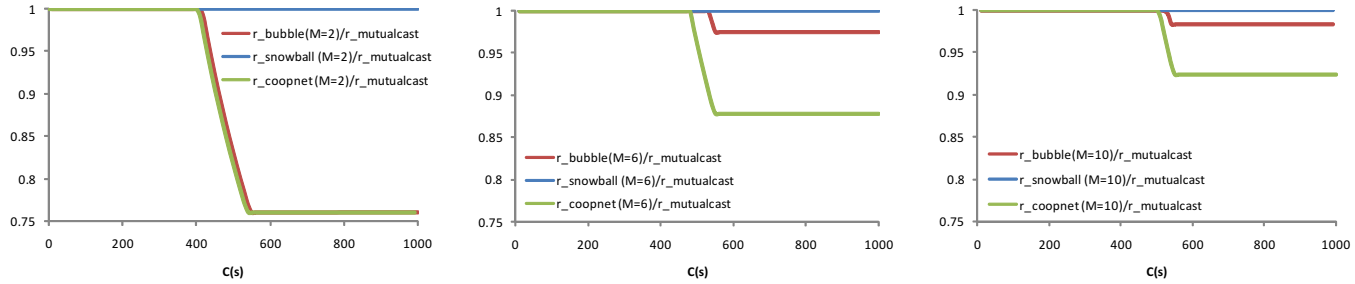


Fig. 9. Streaming rate ratio (w.r.t. Mutualcast) vs. source uplink $C(s)$ for different algorithms; $N = 100,000$ nodes, degree bound $M = 2, 6, 10$.

algorithm (i.e., it could be as high as N). However, simulations on the P2P topologies show that the average degree of a node is about twice that of the degree bound M, as summarized in Fig. 10. This is comparable to the total degree in the CoopNet/SplitStream scheme. Thus, the outdegree parameter M is useful in controlling the total node degree in the Bubble algorithm.

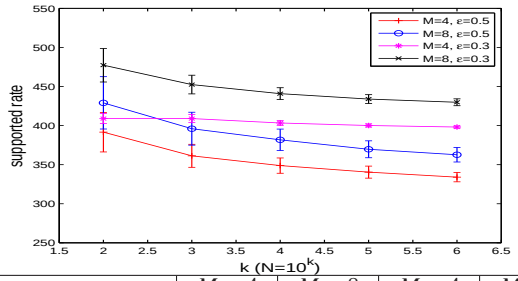
B. Cluster-Tree Algorithm

We now study the performance of the Cluster-Tree scheme. We first study the validity of the asymptotic streaming rate bounds using an empirical peer uplink distribution. Next, we evaluate the ability of the scheme to provide a high sustained streaming rate on a real-world online video viewing trace.

1) Evaluation of Asymptotic Bounds: We consider networks with number of nodes N ranging from 100 to 10^6, and choose the node uplink capacities randomly from a distribution in Table V that is obtained from real peer usage data [21]. We compute the mean μ and variance σ^2 of the upload capacities. In our experiments, we set $\alpha = 1$, where α is the parameter in in (18) and (19).

The supported rate shown in Theorem 2 is an asymptotic result. We would like to study the actual supported rate of the Cluster-Tree algorithm for finite N in the first experiment. For each value of N, we fix ϵ and M, and generate a set of N node upload capacities independently from the distribution in Table V. We then run the Cluster-Tree algorithm to build clusters and cross-cluster trees, and find the supported rate. We carry out 1000 runs of such experiments, collect 1000 samples of the supported rate, and compute their average and standard deviation. We vary N from 100 to 10^6 to get a curve of N vs. supported rates for a set of ϵ and M. We repeat the experiments for $\epsilon = 0.5, 0.3$ and $M = 4, 8$. All resulting curves are shown in Fig. 11. We observe that the achieved rate is significantly higher than the theoretically predicted rate from Theorem 2; the latter is shown in the table in Fig. 11.

Two additional observations can be drawn from Fig. 11. First, for fixed M and ϵ, the supported rate of the Cluster-Tree scheme decreases as N increases. This is because when M and ϵ are fixed, the cluster size is on the order of $\ln N$. Thus, large N means more clusters, and the chance for some cluster-rate X_i to be small increases, pushing down the overall minimum $\underline{X}$. In the extreme case where N is infinity, we almost surely will always have one small cluster-rate. The second observation is that for a fixed N and M, small ϵ leads to small



	$M=4$ $\epsilon=0.5$	$M=8$ $\epsilon=0.5$	$M=4$ $\epsilon=0.3$	$M=8$ $\epsilon=0.3$
$\frac{M-1}{M}(1-\epsilon)\mu$ (kbps)	205	239	286	333

Fig. 11. Supported rates of the Cluster-Tree scheme vs. the number of peers N, for different M and ϵ.

variation in supported rates. This is because as ϵ decreases, the Cluster-Tree algorithm builds larger clusters. The cluster rates X_i are more concentrated around their mean μ, and so does the minimum one $\underline{X}$. In the extreme case where ϵ is almost zero, both X_i's and $\underline{X}$ equal μ and do not vary.

In the second experiment, we study throughput and delay trade-off in the Cluster-Tree algorithm for $N = 10^6$, by varying the cluster size. According to (18), for fixed N, we can vary the cluster size by changing ϵ. Hence, we get different supported rates as we change the cluster size. For fixed M, the playback delay, according to (21), is determined by the cluster size only. We assume the video packet size is 1 KB and the propagation delay between peers is 20 ms. In Fig. 12, we plot the throughput and delay as a function of cluster size $\underline{n}$ for $M = 8$ and $M = 16$.

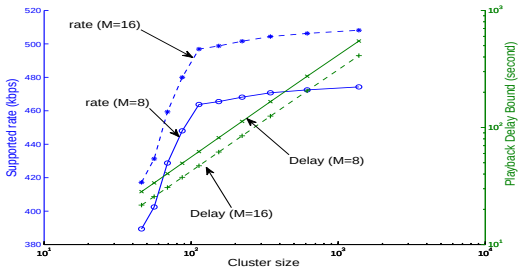


Fig. 12. Supported rate (Kbps) vs. streaming delay (sec) of the Cluster-Tree scheme, as a function of the cluster size.

As seen from Fig. 12, for fixed M, the supported rate increases rapidly as the cluster size increases and quickly saturates. This is expected as ϵ decreases and the cluster size increases. The streaming delay, on the other hand, increases

dramatically as the cluster size increases. There is clearly a trade-off between supported rate and delay, as discussed in Section IV-B. Intuitively, one should pick the cluster size that leads to high supported rate and low playback delay. For instance, according to Fig. 12, in a P2P streaming system with 10^6 peers and $M = 8$, having 70-nodes in all clusters can achieve a streaming rate of 420 Kbps, and at the same time, a streaming delay of 500 seconds, i.e., the largest delay from the source to any receiver is at most 500 sec.

2) Evaluation on Online Video Viewing Trace: In the third experiment, we evaluate the ability of Cluster-Tree algorithm to provider a sustained streaming rate as peers join over time. For this purpose, we use an MSN VoD (Video-on-Demand) trace from [21] – this corresponds to the most popular video on MSN during the measurement period going back to 2007. A total of about 1.3 million peers appear in the trace. Because our schemes are designed for *live streaming* (vs. VoD), we use the trace to model peer arrivals who stay in the system for the rest of the duration of the simulation. Hence, the number of peers in our simulation grow to about 1.3 million over time.

Forming Clusters: The clusters in our Cluster-Tree algorithm require full-mesh communication between them. If peers in a given cluster come from all over the globe, then Internet links would become bottlenecks for communication between them; moreover, criss-crossing paths around the globe within a cluster would lead to inefficient use of Internet bandwidth. For these reasons, we restrict clusters to consist of peers from the same Autonomous System (AS). Such a cluster formation scheme is also ISP-friendly and minimizes cross-ISP peering traffic. With clusters now following AS (and geographic) locality, the asymptotic rate bounds established for Cluster-Tree scheme may not hold, since uplink capacities in the same AS could be correlated due to similar consumer broadband plans provided by the ISP. The main objective of this experiment is to evaluate the impact of this on the streaming rate provide by Cluster-Tree algorithm and propose modifications to improve the algorithm in a practical real-world setting.

Sustained Streaming Rate: Our initial run of the Cluster-Tree algorithm on the MSN video trace with clusters formed with AS domains as described above gives a low sustained streaming rate of about 128 Kbps. This is because a small number of low uplink capacity peers form clusters with low average uplink capacities, and these clusters become the bottleneck for the streaming rate. If we remove these small fraction of very low uplink capacity peers from consideration, the streaming rate delivered by uplink capacity peers could improve significantly. To understand this effect, we demonstrate, in Fig. 13, the streaming rate as a function of the fraction of low uplink capacity peers removed from consideration. We observe that by removing just 0.3% of the 1.3 million peers, the streaming rate is increased to 300Kbps.

The proposal then is to use a different delivery mechanism (e.g., direct from content server) or a separate instance of the Cluster-Tree scheme for this small fraction of ultra-low uplink capacity peers. To validate that this works, we demonstrate, in Fig. 13, the sustained streaming rate in the Cluster-Tree scheme as peers join over time, when a small fraction (we call

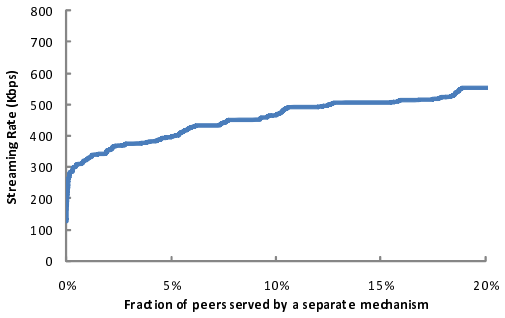


Fig. 13. Streaming rate (Kbps) for Cluster-Tree scheme as a function of the fraction of peers (the lowest uplink capacity ones) served by a separate delivery mechanism. (MSN online video trace used for simulation.)

this fraction a cutoff percentage) of the peer population (the lowest uplink capacity peers) are served through a separate delivery mechanism. We further demonstrate, in Fig. 14, the streaming rates as functions of peer number for various cutoff percentages, and compare them with the theoretical maximum streaming rate (computed as the global average uplink capacity μ). From Fig. 14, a higher cutoff percentage gives a higher streaming rate.

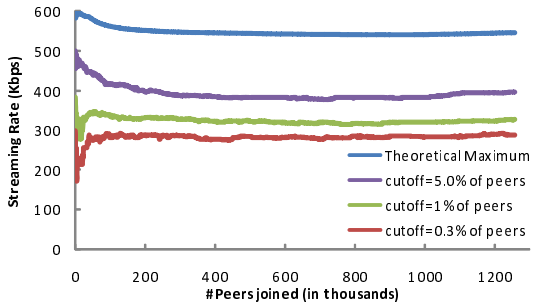


Fig. 14. Sustained streaming rate (Kbps) for Cluster-Tree scheme as peers arrive over time. Different fixed fractions of peers (the lowest uplink capacity ones) are served by a separate delivery mechanism on a continuing basis. Theoretical maximum streaming rate is also shown. (MSN online video trace used for simulation.)

In summary, our evaluations on a real-world video viewing trace show that the Cluster-Tree algorithm can sustain a high streaming rate even when clusters are formed by geographical (AS) proximity, provided a separate delivery mechanism is used for the ultra-low uplink capacity peers.

VI. Conclusion and Future Work

In prior research works, P2P streaming capacity has only been studied without node degree bounds. In this paper, we propose two algorithms, Bubble algorithm for arbitrary node out-degree bound, and Cluster-Tree algorithm for node degree bound of $O(\ln N)$. Bubble algorithm provides a deterministic worst case performance bound of half of the streaming capacity, while Cluster-Tree algorithm achieves near-optimality, through a peering construction method that can be readily implemented in practical P2P control plane. A combination of combinatorial optimization, Central Limit Theorem, and realistic simulation is used to demonstrate the algorithms. Both analysis and numerical experiments show that peering in a locally dense and globally sparse manner achieves near-optimal

streaming rate if the degree bound is at least logarithmic in network size.

We compare our two algorithms in Table V.

There are several future directions. First, we can extend our Cluster-Tree experiments from static networks to dynamic networks with peer churns, and investigate its performance. Also, when we handle the peer join and leave for the Cluster-Tree algorithm, we assume that the tracker only knows the information on the size of clusters, and does not know the bandwidth of the joining/leaving peers peers. If the tracker is able to acquire such information, we can modify our clustering policy to further improve the supported streaming rate. Finally, delay minimization and the corresponding peering construction algorithms remain under-explored.

Acknowledgement

The authors would like to thank David Johnson, Chandra Nair, Anke van Zulyen, and Frans Schalekamp for their helpful discussions.

References

[1] J. Li, P. A. Chou, and C. Zhang, "Mutualcast: an efficient mechanism for one-to-many content distribution," in *Proc. ACM SIGCOMM ASIA Workshop*, Beijing, China, April 2005.
[2] S. Liu, R. Zhang, J. Jiang, J. Rexford, and M. Chiang, "Performance bound in peer-assisted live streaming," in *Proc. ACM Sigmetrics*, Annapolis, MA, June 2008.
[3] S. Liu, M. Chiang, S. Sengupta, J. Li, and P. A. Chou, "Streaming capacity for p2p with degree bound," in *Proc. Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, September 2008.
[4] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in cooperative environments," in *Proc. ACM symposium on Operating systems principles*, New York, NY, October 2003.
[5] V. Padmanabhan and K. Sripanidkulchai, "The case for cooperative networking," in *Proceedings of IPTPS02*, Cambridge, MA, March 2002.
[6] Y. Cui, B. Li, and K. Nahrstedt, "On achieving optimized capacity utilization in application overlay networks with multiple competing sessions," *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, June 2004.
[7] S. Sengupta, S. Liu, M. Chen, M. Chiang, J. Li, and P. A. Chou, "Streaming capacity in p2p networks with topology constraints," submitted to IEEE Trans. on Information Theory, 2009.
[8] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," in *Proc. IEEE Infocom 2007*, Anchorage, AL, April 2007.
[9] T. Nguyen, K. Kolazhi, R. Kamath, S. Cheung, and D. Tran, "Efficient multimedia distribution in source constraint networks," *IEEE Trans. on Multimedia*, vol. 10, no. 3, April 2008.
[10] D. M. Chiu, R. W. Yeung, J. Huang, and B. Fan, "Can network coding help in p2p networks?" in *Proc. of IEEE NetCod*, Boston, MA, USA, April 2006.
[11] L. Massoulie, A. Twigg, G. Gkantsidis, and P. Rodriguez, "Randomized Decentralized Broadcasting Algorithms," in *Proc. IEEE INFOCOM*, Anchorage, AL, May 2007.
[12] Y. Guo, C. Liang, and Y. Liu, "dHCPS: decentralized hierarchically clustered p2p video streaming," in *Proc. ACM International conference on Content-based Image and Video Retrieval*, New York, NY, July 2008.
[13] F. López-Fuentes and E. Steinbach, "Hierarchical collaborative multicast," in *Proc. the 15th International Conference on Multimedia*, Augsburg, Germany, September 2007.
[14] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. Chou, "Utility maximization in peer-to-peer systems," in *Proc. ACM SIGMETRICS*, Annapolis, MD, June 2008.
[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, 1979.
[16] PPLive. http://www.pplive.com/.
[17] PPStream. http://www.ppstream.com/.
[18] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *Multimedia, IEEE Transactions on*, vol. 9, no. 8, pp. 1672–1687, 2007.
[19] T. Bonaldy, L. Massoulie, F. Mathieuy, D. Perinoy, and A. Twigg, "Epidemic live streaming: Optimal performance trade-offs," in *ACM Sigmetrics*, Annapolis, MD, June 2008.
[20] Y. Liu, "Delay bounds of peer-to-peer video streaming," nYU/Polytechnic Institute Technical Report, Jun. 21, 2009.
[21] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" in *Proc. ACM SIGCOMM*, Kyoto, Japan, August 2007.

VII. Appendix

A. Proof of Proposition 1

We first prove Proposition 1.

Proof: It has been shown in [15] that the DEGREE-BOUNDED-SUBGRAPH problem of determining the existence of a subgraph of G with node degree bound $D \geq 2$ is NP-Complete. Any solution to the problem in (1) also gives a subgraph of G with node degree bound $D \geq 2$, which is a solution to the NP-Complete DEGREE-BOUNDED-SUBGRAPH problem. As such, the problem in (1) for general graph must also be NP-Complete. ■

B. Proof of Results in Section 3

In this section, we prove all results in Section III, i.e., Lemma 1, Proposition 2, Theorem 1, and Proposition 3.

We first prove Lemma 1.

Proof: We first prove $r_s^* \geq r_b^*$. This is straightforward, since otherwise, if server capacity is $S = r_b^*$, then we have $r_b(S) = S > r_s(S)$, which violate the condition of $r_s(S) \geq r_b(S), \forall S > 0$.

We then prove (11). The first inequality comes from the facts that snowball is optimal for tree degree bound and thus superior to optimal node degree bound. The second inequality is visually seen from Figure 5. The third inequality comes from the monotonicity of $r_b(S)$, and the last equality is from the definition of r^*. ■

Before we prove Proposition 2, we need to introduce another lemma. Recall that Snowball algorithm has a good property of being *most efficient*, i.e., after it stops, all nodes either are exhausted or trimmed with the largest possible rate. Define *effective capacity* of node v to be the minimum of its capacity $C(v)$ and Mr, then Snowball algorithms always consumes all peers' effective capacities and yield the optimal streaming rate under tree degree bound. For the most efficient algorithms, we have the following result:

Lemma 3: Suppose we have N peers and two tree construction algorithms A and B, both construct initial trees unless not possible, both keep the tree degree bounded by M, and both are most efficient. Suppose the critical rates for the two algorithms are r_A^* and r_B^*, and the largest index nodes left unexhausted are j_A and j_B, respectively. Then, the following are true:

1) For algorithm A, the unexhausted nodes are nodes 1 to j_A; similarly for algorithm B, the unexhausted nodes are nodes 1 to j_B.
2) $j_A = j_B$ and $r_A = r_B$.
3) Both algorithm A and algorithm B achieve streaming capacity under tree degree bound.

Lemma 3 says that two most efficient initial tree construction algorithms are equivalent in terms of bandwidth utilization and streaming rate, and both are optimal algorithms.

We next prove Lemma 3.

Proof: We first prove 1). For algorithm A, if node j is left unexhausted, then $Mr < C_j$. For all $i < j$, $C_i \geq C_j > Mr$,

TABLE V
COMPARISON OF THE BUBBLE AND CLUSTER-TREE ALGORITHMS. "W.H.P." IS SHORT OF "WITH HIGH PROBABILITY".

	Type of degree bound	Transport Protocol	Scale of Network	Scale of Degree Bound	Optimality	Practicality
Bubble	node out-degree	UDP	arbitrary	arbitrary	1/2 guaranteed	No
Cluster-Tree	node total degree	TCP/UDP	large-scale	$O(\ln N)$	$1-\epsilon$ w.h.p.	Yes

so all nodes $i < j$ are unexhausted also. Therefore, 1) holds for algorithm A, and similar reasoning applies to algorithm B.

We then prove 2) using contradiction. Suppose $j_A > j_B$. Since

$$C_{j_B} > Mr_B^* \geq C_{j_B+1} \geq C_{j_A} > Mr_A^*,$$

we know that $r_B^* > r_A^*$.

On the other hand, from [2], [3], we have the following formulas:

$$r_A^* = \frac{\sum_{k>j_A} C_k}{N-1-Mj_A} \tag{22}$$

$$r_B^* = \frac{\sum_{k>j_B} C_k}{N-1-Mj_B} \tag{23}$$

Since any node $k > j_B$, it is exhausted, and we know $C_k \leq Mr_B^*, \forall k > j_B$, and this applies to $k = j_B + 1$ to j_A. Then,

$$\begin{array}{rcl} r_B^* & = & \frac{1}{N-1-Mj_B}\left(C_{j_B+1}+\cdots+C_{j_A}+\sum_{k>j_A}C_k\right) \\ & \leq & \frac{1}{N-1-Mj_B}\left(Mr_B^*+\cdots+Mr_B^*+\sum_{k>j_A}C_k\right) \\ & = & \frac{1}{N-1-Mj_B-M(j_B-j_A)}\sum_{k>j_A}C_k \\ & = & \frac{1}{N-1-Mj_A}\sum_{k>j_A}C_k = r_A^* \end{array} \tag{24}$$

Then, we arrive at contradiction. Similarly, $j_A < j_B$ also leads to contradiction, and therefore, the only possibility is $j_A = j_B$, and from (22), $r_A^* = r_B^*$ also.

WE finally prove 3). Since Snowball algorithm is most efficient and is optimal, all most efficient algorithms are equivalent to Snowball, and therefore they are optimal under tree degree bound. ∎

With Lemma 1 and 3 at hands, we can prove Proposition 2.

Proof: We start from the case that all the I internal nodes have M children in initial trees, i.e., $N = 1+IM$. For this case, all internal nodes are trimmed with the same rate.

For both snowball algorithm and bubble algorithm, they try to use initial trees as long as it is possible. When that is not doable, suppose the remaining bandwidths of node v is $\tilde{C}_v$, and there are J_s (respectively, J_b) nodes left unexhausted (positive remaining bandwidth) when the snowball (respectively, bubble) algorithm ends. Then, both $J_s \leq I-1$ and $J_b \leq I-1$. For snowball algorithm, the J_s nodes are always the first J_s nodes indexed from 1 to J_s. For bubble algorithm, the J_b nodes could be discontinuous. Suppose a node $j < I$ is left unexhausted, then this node must be an internal node and have outgoing degree M in any constructed initial trees. Since $C_i \geq C_j, \forall i \leq j$, we know that all the nodes from 1 to $j-1$ are left unexhausted also. Suppose there are J_1 unexhausted nodes whose indexes are less than I, then there are $J_b - J_1$ unexhausted nodes whose indexes are larger than I (the I-the node is exhausted at the first iteration). It is clear that $J_1 \geq J_s$, since $C_{J_s} \geq Mr_s^* \geq Mr_s^* > C_{J_1+1}$, which leads to $J_s < J_1 + 1$.

We divide the index set $\{1, 2, \cdots, N\}$ to three sets:

$$\left\{\begin{array}{rcl} A & := & [1, J_1] \\ B & := & [J_1+1, I] \\ C & := & \{i : I < i \leq N, \tilde{C}_i > 0\} \\ D & := & \{i : I < i \leq N, \tilde{C}_i = 0\} \end{array}\right. \tag{25}$$

Then, we know that $|A| = J_1$, $|B| = I - J_1$, $|C| \leq I - 1 - J_1 < |B|$ and we have

$$\begin{array}{rcl} r_b^* & = & \frac{1}{N-1}\left(\sum_{v\in A}(C_v-\tilde{C}_v)+\sum_{v\in B}C_v+\sum_{v\in C}(C_v-\tilde{C}_v)+\sum_{v\in D}C_v\right) \\ & = & \frac{1}{N-1}\left(J_1 r_b^*+\sum_{v\in B}C_v+\sum_{v\in C}(C_v-\tilde{C}_v)+\sum_{v\in D}C_v\right) \\ & = & \frac{1}{N-1-J_1}\left(\sum_{v\in B}C_v+\sum_{v\in C}(C_v-\tilde{C}_v)+\sum_{v\in D}C_v\right) \\ & \geq & \frac{1}{N-1-J_1}\sum_{v\in B}C_v \end{array}$$

From Lemma 3, we know that if $C = \emptyset$, then bubble algorithm achieves the same performance as the snowball algorithm.

If $C \neq \emptyset$, we see that there are more unexhausted nodes for bubble algorithm than the snowball algorithm, and therefore, the $r(U_s)$ curve for bubble algorithms is below that for the snowball algorithm. Consider why we have non-empty C set, that is because there are less than I nodes left unexhausted and no initial tree could be further constructed. Suppose we can reallocate the remaining bandwidth of nodes in set C to all nodes from $I+1$ to N, and suppose we relax the degree constraint to tree degree bound, then we can exhaust all nodes in set C. Suppose we have supported an extra rate r_e in this extra step, then clearly, the total rate we have supported is $r_b^* + r_e$, and we have

$$r_e \leq \frac{\sum_{vinC}\tilde{C}_v}{N-1-Mj_1}$$

From Lemma 3, we know that any two algorithms reach the same result if they both use up all nodes' effective capacities, and therefore, we know that

$$r_b^* + r_e = r_s^*$$

Furthermore, since $|B| > |C|$ and $C_v > C_u, \forall v \in B, u \in C$, we know that

$$r_b^* \geq \frac{\sum_{v\in B}C_v}{N-1-J_1} > \frac{\sum_{v\in C}C_v}{N-1-J_1} \geq \frac{I}{I-1}r_e \tag{26}$$

which means that $r_b^* \geq r_s^* I/(2I-1)$.

If $N \neq IM+1$, which means $N < IM+1$, then the last internal node has less than M children and the I nodes are not trimmed with the same rate. For that case, we can add several fake nodes as the children to the last internal node, so that it also has M children (real or fake), and all nodes are trimmed with the same speed now. Then, the analysis above still applies, and we need to scale down the Bubble performance by a factor. It is shown in a longer version of this paper that, this scale down factor does not affect the worst

case ratio.

From Figure 5 and Lemma 1, $r_b(S)/r_s(S) \geq r'_b/r^*_s$. It is easy to find that, at the worst case, $r'_b = r^*_b + (r^*_s - r^*_b)/(1+M)$. Plugging (26), we have proved (12). ∎

The proof of Theorem 1 is directly derivable from Lemma 1 and Proposition 2, noting the fact that

$$\frac{I}{2I-1} + \frac{I-1}{(2I-1)(1+M)} > \frac{1}{2}, \forall N, M\,.$$

C. Proof of Proposition 3

We finally prove Proposition 3.

Proof: For the case of homogeneous receivers, interior nodes of all Bubble trees are disjoint, since all interior nodes in a tree use up their capacities simultaneously, and are leaf nodes in all the rest trees.

In a Bubble tree with tree degree M, there are one root and $\lceil \frac{N-1}{M} \rceil$ interior nodes. Given N receivers, the total number of degree-d interior-node-disjoint Bubble trees can be constructed is $\lfloor N/\left(\lceil \frac{N-1}{M} \rceil\right) \rfloor$.

For $N > M+1$, there exists some integers $k \geq 1$ and $0 \leq l < M$ such that $N-1 = kM + l$, we have $k \leq \lceil \frac{N-1}{M} \rceil \leq k+1$; hence,

$$\left\lfloor \frac{N}{k+1} \right\rfloor \leq \left\lfloor \frac{N}{\lceil \frac{N-1}{M} \rceil} \right\rfloor \leq \left\lfloor \frac{N}{k} \right\rfloor. \tag{27}$$

We also have

$$\frac{N}{k} = M + \frac{l+1}{k} < M + \frac{l+1}{M+1} < M+1,$$

and

$$\frac{N}{k+1} \geq \frac{k}{k+1}M \geq \frac{M+1}{M+2}M \geq \frac{M-1}{M}M = M-1.$$

Combining the above two observations into (27), we get

$$M-1 \leq \left\lfloor \frac{N}{\lceil \frac{N-1}{M} \rceil} \right\rfloor \leq M.$$

Since a node at most has M distinct parents across M trees and has at most M children when it is a parent node, overall the degree of the node is upper bounded by $2M$.

The supported rate of each Bubble tree is $\frac{C}{M}$, the total number of trees is at least $M-1$; hence, the aggregate rate supported by this set of Bubble trees is at least $\frac{M-1}{M}C$. ∎

D. Proof of Results in Section 4

The proof of Proposition 4 is straightforward, since the total rate needed is $X_i n_i$ and the total available capacity is also $X_i n_i$; and if we get outside rate at X_i, then we have X_i extra and can be used to serve outside.

We first prove Proposition 5.

Proof: Since X_i is the average upload capacity of cluster G_i, there must exist a node whose upload capacity exceeds X_i. Let one of this node be the node v_1. If $C_{v_1} \geq 2X_i$, then we are done. If $2X_i > C_{v_1} \geq X_i$, we now prove that there must exist another node v_2 so that $C_{v_1} + C_{v_2} \geq 2X_i$. Suppose this is not true. Then for all $v \in G_i - \{v_1\}$, $C_v < 2X_i - C_{v_1}$. But this implies

$$\begin{aligned} X_i = \frac{1}{n_i}\sum_{v \in G_i} C_v &< \frac{n_i - 1}{n_i}(2X_i - C_{v_1}) + \frac{1}{n_i}C_{v_1} \\ &= X_i + \frac{n_i - 2}{n_i}(X_i - C_{v_1}) \\ &\leq X_i, \end{aligned}$$

which leads to a contradiction. ∎

We then prove Theorem 2

Proof: We have

$$\begin{aligned} P(\underline{X} < (1-\epsilon)\mu) &= P\left(\min_{i=1,\ldots,K} X_i < (1-\epsilon)\mu\right) \\ \text{(union bound)} \quad &\leq \sum_{i=1}^{K} P(X_i < (1-\epsilon)\mu) \\ &= \sum_{i=1}^{K} P\left(\frac{X_i - \mu}{\sigma/\sqrt{n_i}} < -\sqrt{n_i}\frac{\epsilon\mu}{\sigma}\right) \\ (X_i \text{ is Gaussian}) \quad &< \sum_{i=1}^{K} \exp\left(-\frac{1}{2}n_i\left(\frac{\beta\mu}{\sigma}\right)^2\right) \\ &\leq \frac{N}{\underline{n}} \exp\left(-\frac{\underline{n}}{2}\left(\frac{\epsilon\mu}{\sigma}\right)^2\right). \end{aligned}$$

Since the peer nodes are distributed into clusters evenly, if $\underline{n} = \left\lceil \frac{2(1+\alpha)\sigma^2}{\epsilon^2\mu^2} \ln N \right\rceil$, then

$$P(\underline{X} < (1-\epsilon)\mu) < \frac{1}{2(1+\alpha)}\left(\frac{\epsilon\mu}{\sigma}\right)^2 \frac{1}{N^\alpha \ln N},$$

which diminishes to zero as N goes to infinity. ∎

We finally prove Lemma 2.

Proof: The proof is very similar to the one showing 4-PARTITION is NP-Complete in [15, Theorem 4.3]. For the sake of completeness, we write down the proof as follows.

We transform an instance of 4-PARTITION problem to a restricted version of the average-bin-weight problem, with certain ball weights. As this particular instance of 4-PARTITION has been shown to be as hard as 3-DIMENSIONAL MATCHING (3DM) and is NP-complete, so does the average-bin-weight problem even when restricted to this instance.

Let $X = \{x_1, x_2, \ldots, x_q\}$, $Y = \{y_1, y_2, \ldots, y_q\}$, $Z = \{z_1, z_2, \ldots, z_q\}$, and $\mathcal{M} \subseteq W \times X \times Y$ denote arbitrary instance of 3DM.

Our corresponding instance has $n = 4|\mathcal{M}|$ balls, one for each occurrence of a member of $X \cup Y \cup Z$, and one for each triple in $\mathcal{M}$.

The elements corresponding to a particular $p \in X \cup Y \cup Z$ will be denoted by $p[1], p[2], \ldots p[N(p)]$, where $N(p)$ denotes the number of triples from $\mathcal{M}$ in which p occurs. The weights are defined as follows:

$$w(x_i[l]) = \begin{cases} 10(32q)^4 + i \cdot 32q + 1, & 1 \leq i \leq q, l = 1; \\ 11(32q)^4 + i \cdot 32q + 1, & 1 \leq i \leq q, 2 \leq l \leq N(x_i). \end{cases}$$

$$w(y_j[l]) = \begin{cases} 10(32q)^4 + j(32q)^2 + 2, & 1 \leq j \leq q, l = 1; \\ 11(32q)^4 + j(32q)^2 + 2, & 1 \leq j \leq q, 2 \leq l \leq N(y_j). \end{cases}$$

$$w(z_k[l]) = \begin{cases} 10(32q)^4 + k(32q)^3 + 4, & 1 \leq k \leq q, l = 1; \\ 8(32q)^4 + k(32q)^3 + 4, & 1 \leq k \leq q, 2 \leq l \leq N(z_k). \end{cases}$$

We assign the single ball corresponding to a particular triple $(x_i, y_j, z_k) \in \mathcal{M}$, denoted by v, the weight as follows

$$w(v) = 10(32q)^4 - k(32q)^3 - j(32q)^2 - i(32q) + 8.$$

At the end we have $n = 4|\mathcal{M}|$ balls, each with the weight assigned according to the above rule. It can be verified that the total ball weights is $B \cdot |\mathcal{M}|$, where $B = 40(32q)^4 + 15n^3 + 15$.

Now consider the following two problems

- **A 4-PARTITION problem:** Can these n balls be distributed into $|\mathcal{M}|$ bins so that every bin has exactly 4 balls and the total ball weight of every bin is exactly B?
- **An average-bin-weight assignment problem:** Can these n balls be distributed into $|\mathcal{M}|$ bins so that every bin contains at least one balls and the average ball weight of every bin is exactly $B/4$?

On one hand, the authors in [15, Theorem 4.3] show that the desired 4-partition exists if and only if $\mathcal{M}$ contains a matching. That is, this instance of 4-PARTITION is as hard as 3DM and is NP-Complete.

On the other hand, we argue that the desired 4-partition exists if and only if the average-bin-weight assignment exists. First, clearly any valid 4-partition is also a valid average-bin-weight assignment. Second, in valid average-bin-weight assignment, every bin must contain exactly 4 balls, and thus a valid 4-partition. To see this, notice every ball has integer weight, and the fraction part of $B/4$ is 0.25. For any bin to have average ball weight $B/4$, it must contain at least 4 balls. Since there are totally $4|\mathcal{M}|$ balls and $|\mathcal{M}|$ bins, every bin must have exactly 4 balls to have average ball weight of exactly $B/4$.

As such, this instance of average-bin-weight problem is as hard as 3DM and is NP-Complete. ■