

P2P Streaming Capacity under Node Degree Bound

Shao Liu¹, Minghua Chen², Sudipta Sengupta³, Mung Chiang¹, Jin Li³, and Phil A. Chou³
¹ Princeton University, ² The 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. We first show that the streaming capacity problem under node degree bound is NP-Complete in general. Then, for the case of node out-degree bound, through the construction of a “Bubble algorithm”, we show that the streaming capacity 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 of node total-degree bound, we develop a “Cluster-Tree algorithm” that provides a probabilistic guarantee of achieving a rate close to the maximum rate achieved under no degree bound constraint, when the node degree bound is logarithmic in network size. The effectiveness of these algorithms in approaching the capacity limit is demonstrated in simulations using uplink bandwidth statistics of Internet hosts. 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 Section II, P2P streaming can be modeled as multiple multicast trees superimposed on top of the overlay graph. 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]–[12]. Various practical constraints on peering and overlay graph have since been considered. For example, [3], [4] 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, [2], [5], [12]–[14] 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). As discussed later in Section II-B, bounding node degree is necessary for TCP-based communications, while bounding out-degree is sufficient for UDP-based communications.

In this paper, we study the P2P streaming capacity problem on P2P overlay graph with capacity constraints and degree bounds on *nodes*. Thus it is different from a large body of work focusing on underlay graph with node degree constraints and capacity constraints on *links* (see for instance [2], [15]–[17] and the related work in [5]). In reality, P2P streaming system is usually unstructured, and cannot achieve the streaming capacity. For the optimal scheduling in unstructured P2P system and the performance gap between unstructured practical P2P system and streaming capacity, please see [18]–[21].

In the P2P study, the problem we study has not been pursued even in the case of full mesh graph without helper nodes, due to the challenges introduced by node degree bound. Algorithms such as SplitStream/CoopNet [6], [7], ZIGZAG [22], and PRIME [23] bound node degree but do not provide any streaming rate guarantee. 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. Section III is primarily for theoretical interest.
- 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

This work has been in part supported by NSF grants CNS-0519880 and CNS-0720570. This work is also supported by RGC GRF grant 411008 and 411209, RGC AoE grant on Institute of Network Coding, established under the University Grant Committee of Hong Kong, CUHK MoE-Microsoft Key Laboratory of Human-centric Computing and Interface Technologies, and two gift grants from Microsoft and Cisco. Part of the work is done when Shao Liu visited the Chinese University of Hong Kong, under the support of RGC GRF grant 411008 and 411209.

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

| Classification | References | Graph | Tree Degree | Node Degree | Optimality |
|-----------------------|----------------------------------|---------------------|-------------|--------------------|---------------------------------|
| no degree bound | Mutualcast [1] | unbounded | unbounded | complete, directed | exact optimality |
| no degree bound | Iterative in [2] | general, undirected | unbounded | unbounded | $1 - \epsilon$ approx. |
| per-tree degree bound | Snowball [3], [4] | complete, directed | bounded | unbounded | exact optimality |
| per-tree degree bound | Iterative in [5] | general, directed | bounded | unbounded | $1 - \epsilon$ approx. |
| node out-degree bound | Bubble (Section III) | complete, directed | bounded | bounded, arbitrary | $> 1/2$ approx. |
| node degree bound | CoopNet/SplitStream [6], [7] | complete, directed | bounded | bounded, arbitrary | no optimality |
| node degree bound | Cluster-tree (Section IV) | complete, directed | bounded | bounded, $\log N$ | $1 - \epsilon$ approx. (w.h.p.) |

to achieve near-optimal streaming rate. Section IV is for both theoretical interest and practical application. Performance guarantee of the Cluster-Tree algorithm 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.

- We demonstrate in Section V the effectiveness of our algorithms through simulations using uplink capacity statistics of Internet hosts.

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 . |
| $N = R $ | size of R ; the number of all receiver peers. |
| 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 receivers, divide the stream into multiple substreams, and route one substream per tree. In mesh-based P2P streaming approaches [13], [14], peers dynamically exchange video packets with several of their neighbors. No explicit tree is constructed. Of course, in mesh-based P2P, the trajectory of each packet still forms a spanning tree, originating from the source and reaching each peer exactly once.

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. 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 [2], [5]. 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 is bounded by its uplink capacity C_v , e.g., 384 Kbps for typical ADSL users in the U.S. Similarly, its aggregate incoming rate is bounded by its downlink capacity, e.g., 1.5 Mbps. Similar to most other P2P work [1], [8], [10]–[12], we make the “uplink bottleneck” assumption that node uplinks are the only rate limiting bottlenecks in the network. We also assume a node only stores and forwards content, and thus do not consider network coding.

B. Degree Bounds

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. Bounding node degree also helps to reduce the fanout delay, which is the time a node takes to forward a packet received from its parent(s) to its children.

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. A receiving node can use one single socket to receive packets from all its parents. Hence, the overhead is proportional to the node’s out-degree but independent to the node’s in-degree. Furthermore, fanout delay also depends on the out-degree only.

Similar to [3], [4], 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 $Q(G, D)$ 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 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):

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

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

The streaming capacity is the optimal value of Problem 1.

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

The streaming capacity for the case of $D = \infty$ is recently characterized by a supply-equal-demand argument as follows [1], [8], [10]:

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

It can be achieved by packing one depth-1 tree and N depth-2 trees, called *Mutualcast trees* [1] [12] [8].

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:

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

However, when node degree bounds are imposed, it becomes challenging to either compute or come close to the streaming capacity.

Proposition 1: For general graph G and $D \geq 2$, Problem 1 is NP-Complete.

Proof: It has been shown in [24] that the DEGREE-BOUNDED-SUBGRAPH problem of determining the existence of a connected subgraph of G with node degree bound $D \geq 2$ is NP-Complete. Any solution to the problem in (1) also gives a connected 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. ■

Remarks: This problem can be modeled as a mixed integer linear program (MILP) with continuous variables representing rates on trees and integer 0/1 variables indicating whether a tree is used or not. The node degree bounds would be

modeled using integer variables. However, the straightforward formulation has exponential number of variables (one for each possible tree). Taking the dual could make the number of variables polynomial but the integer variables in the primal make things more complicated. Moreover, obtaining an ILP formulation does not help us solve the problem in an efficient manner. The best of commercial LP solvers like cplex do not give reasonable running times on MILP formulations on graphs beyond few tens of nodes.

Observing the challenges, in the rest of the paper, we seek polynomial time algorithms with performance guarantees. Note that, there are prior works studying the P2P streaming capacity on top of a non-full-mesh overlay graph; see [2] for undirected graph without any degree bound, and [5] 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*.

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 [3], [4], which propose a ‘‘Snowball algorithm’’ that satisfies p.t.d.b. and achieves the exact streaming capacity $\bar{r}_T(C_s, M)$.

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 $\bar{r}_T(C_s, M)$, for $M = D_o$, i.e.,

$$\bar{r}_N(C_s, D_o) \geq \frac{1}{2} \bar{r}_T(C_s, M)|_{M=D_o}, \forall C_s \geq 0, \forall D_o, N \geq 1. \quad (6)$$

Proof: We prove Theorem 1 by constructing a ‘‘Bubble algorithm’’ that satisfies n.o.d.b., and achieves at least half of the capacity achieved by Snowball algorithm, i.e., $\bar{r}_T(C_s, M)$. The proof is relegated to Appendix A. ■

We now briefly describe the Bubble algorithm. 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 a 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 an ‘‘internal tree’’ of node v . Without loss of generality we assume that $C_1 \geq C_2 \geq \dots \geq C_N$.

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 of each node remains unchanged across its trees. 2) Once we construct a tree, we assign a maximum allowed rate to this tree, such that at least one of the internal nodes is exhausted, and we swap the exhausted

internal node(s) with the largest leaf node(s) and construct new trees. 3) The swap must be done 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 an initial tree. We note that Bubble algorithm is a *centralized* algorithm, and thus not a practical P2P streaming solution. We only design it to prove Theorem 1, and it is the first algorithm that satisfies n.o.d.b. while still having 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. 3. 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, such 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, such 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 take 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.

A complete example of how the Bubble algorithm constructs the multi-tree to utilize peer bandwidths is shown in Fig. 2 and Fig. 3. 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 swapped after each iteration are shown in Fig. 2, and the trees constructed in the iterations are shown in Fig. 1.

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 [3], [4], denoted by $r_b(C_s, D_o)$ and $r_s(C_s, M)$, respectively. We first have

$$r_s(C_s, M) = \bar{r}_T(C_s, M) \geq \bar{r}_N(C_s, D_o) \geq r_b(C_s, D_o), \text{ if } M = D_o. \quad (7)$$

Both Snowball and Bubble algorithms use initial trees until not possible. 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

$$\begin{aligned} r_s(C_s, M) &= C_s \text{ if } C_s \leq r_s^*, \text{ and } r_s(C_s, M) < C_s \text{ otherwise,} \\ r_b(C_s, D_o) &= C_s \text{ if } C_s \leq r_b^*, \text{ and } r_b(C_s, D_o) < C_s \text{ otherwise.} \end{aligned} \quad (8)$$

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

Lemma 1: If $D_o = M$, then:

$$\begin{aligned} r_s^* &\geq r_b^* \\ \frac{\bar{r}_N(C_s, D_o)}{\bar{r}_T(C_s, M)} &\geq \frac{r_b(C_s, D_o)}{r_s(C_s, M)} \geq \frac{r_b(r_s^*, D_o)}{r_s(r_s^*, M)} = \frac{r_b^*}{r_s^*} \end{aligned} \quad (9)$$

Proof: Please refer to Appendix A. ■

Inequality (9) and the first inequality in (10) are straightforward from (7), the third inequality in (10) is straightforward from (9), and the last equality in (10) is straightforward from (8). The key result is the second inequality in (10), 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. 4. From Lemma 1, once we bound r_b^*/r_s^* , we bound the ratio between Bubble rate and Snowball rate, and thus bound the ratio between n.o.d.b. 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

$$\frac{r_b^*}{r_s^*} \geq \frac{I}{2I - 1}, \quad (11)$$

$$\frac{r_b(C_s, D_o)}{r_s(C_s, M)} \geq \frac{I}{2I - 1} + \frac{I - 1}{(2I - 1)(1 + M)}. \quad (12)$$

Proof: Please refer to Appendix A. ■

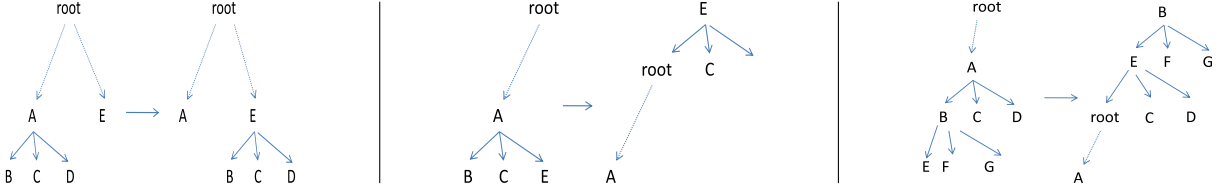


Fig. 1. The swap of an exhausted internal node and a leaf node. From left to right, case 1, case 2 and case 3. Case 1: A is not an ancestor of E. Case 2: A is the parent of E. Case 3: A is an ancestor, but not the parent, of E.

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_i | 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 |
| <u>10</u> | <u>5</u> | <u>(0)</u> | <u>(8)</u> | 7 | 6 | 5 | 4 | 3 | 1.5 | 5/3 | 15/3 | 1 | 15/3 |
| <u>5</u> | <u>(0)</u> | 0 | <u>3</u> | <u>(7)</u> | 6 | 5 | 4 | 3 | 1.5 | 3/3 | 18/3 | 1 | 18/3 |
| <u>2</u> | 0 | 0 | <u>(0)</u> | <u>4</u> | <u>(6)</u> | 5 | 4 | 3 | 1.5 | 2/3 | 20/3 | 1 | 20/3 |
| <u>(0)</u> | 0 | 0 | 0 | <u>2</u> | <u>4</u> | <u>(5)</u> | 4 | 3 | 1.5 | 2/3 | 22/3 | 1 | 22/3 |
| 0 | 0 | 0 | 0 | <u>(0)</u> | <u>2</u> | <u>3</u> | <u>(4)</u> | 3 | 1.5 | 2/3 | 24/3 | 1 | 24/3 |
| 0 | 0 | 0 | 0 | 0 | <u>(0)</u> | <u>1</u> | <u>2</u> | <u>(3)</u> | 1.5 | 1/3 | 25/3 | 1 | 25/3 |
| 0 | 0 | 0 | 0 | 0 | 0 | <u>(0)</u> | <u>1</u> | <u>2</u> | <u>(1.5)</u> | 1/3 | 26/3 | 1 | 26/3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | <u>1</u> | <u>0.5</u> | 1/6 | 26.5/3 | 4 | 28/3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | <u>0.5</u> | 0 | 1/6 | 27/3 | 7 | 31.5/3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | |

Fig. 2. An 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 symbol (-) means the node is to be swapped ((0) means the node is the new formed bubble from the previous iteration).

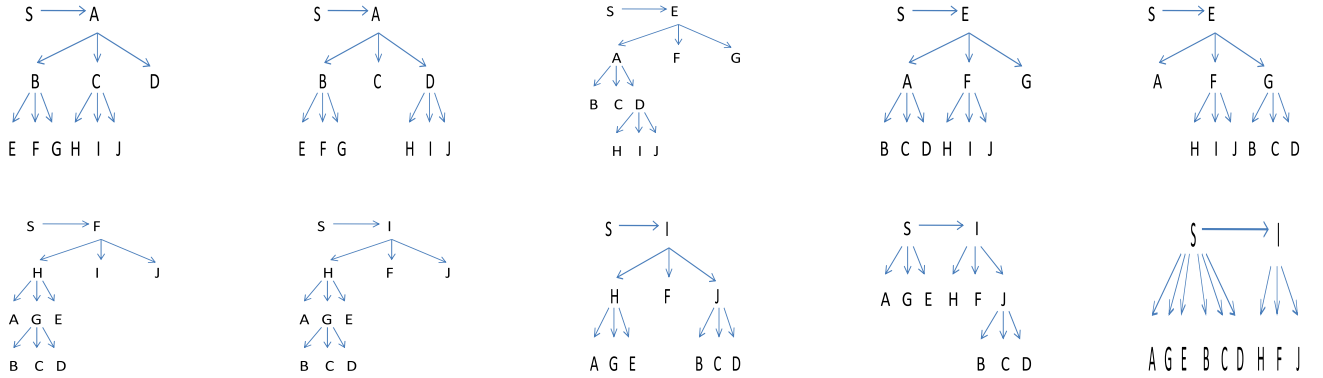


Fig. 3. Constructed trees in all iterations for the example in Fig. 2. The first 8 trees are initial trees. After iteration 8, there are less than $I = 3$ peers left unexhausted, and Bubble algorithms stops. Then server takes care of the children of all exhausted internal nodes from then on.

The inequality in (11) concludes that the ratio of the two critical rates is at least $1/2$, and (12) comes from (11).

From Lemma 1 and Proposition 2, we have:

$$\bar{r}_N(C_s, D_o) \geq r_b(C_s, D_o) \geq \frac{1}{2} \bar{r}_T(C_s, M)|_{M=D_o} \quad (13)$$

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

Equation (13) 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 (14) 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 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 worst 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, $r_b^*/r_s^* \rightarrow 1/2$ and $r_b(C_s, D_o)/\bar{r}_N(C_s, D_o) \rightarrow 1/2$ as both N and M approaches to infinity. Although $1/2$ is the worst case ratio, we will show in Section V that the ratio of $r_b(C_s, D_o)/\bar{r}_N(C_s, \infty)$ is actually very close to 1 for most realistic peer bandwidth distributions.

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. 4.

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.

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:

- 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 $(1 - 1/M)C$.

Proof: Please refer to Appendix A. ■

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 [6] and CoopNet [7]. As such, Bubble trees share the same properties as SplitStream and CoopNet trees, such as robustness to peer churn and low delay delivery.

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. In contrast, in this section, we study the streaming capacity problem under the more difficult node (total) degree bound (**n.d.b.**), and provide a *probabilistic* performance guarantee by developing a Cluster-Tree algorithm.

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. 5. The general idea of clustering in large scale P2P is of course not new, and what the rest of this section does is to develop a particular way to combine clustering with peering to attain streaming capacity under degree bound.

A. Cluster-Tree Algorithm Description

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, \dots, 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 $\underline{n} = \min_{1 \leq i \leq K} n_i$ and $\underline{X} = \min_{1 \leq i \leq K} X_i$.

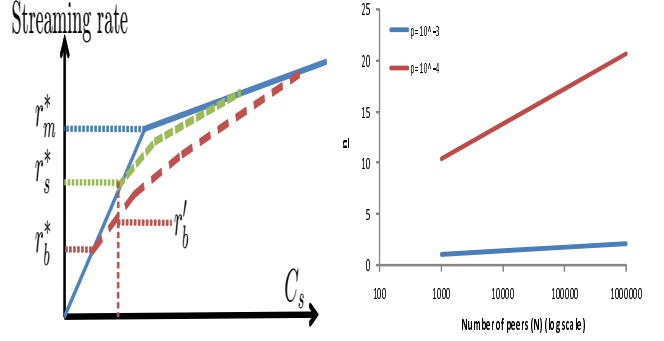


Fig. 4. Plots illustrating Theorem 1 and Theorem 2. Left: The critical rates and streaming capacity functions of Bubble (bottom red curve), Snowball (middle green curve) and Mutualcast (top blue curve) algorithms. r_m^* , r_s^* , r_b^* are the corresponding critical rates. r_b^* is the streaming rate 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_s^* . 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 $(2(1 + \alpha)\sigma^2)/(\epsilon^2\mu^2)$ is the smallest value for (18) to hold for all N from 1000 to 1000,000. It shows that a cluster size of below 25 is enough to guarantee a low outage probability for system size up to one million.

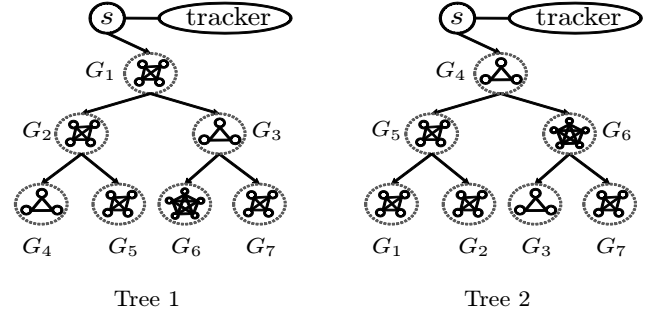


Fig. 5. In this example, the tracker groups the peers into 7 clusters. The server s forms two interior-node-disjoint trees, each having tree degree 2, to distribute video to all clusters. Within each cluster, the peers form a full-mesh and locally broadcast the video among themselves. There are not enough 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.

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 .

Proof: Please refer to Appendix B. ■

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 $C_{v_1} \geq 2X_i$, or there exists two nodes $v_1, v_2 \in G_i$ such that $C_{v_1} + C_{v_2} \geq 2X_i$.

Proof: Please refer to Appendix B. ■

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).

The analysis for the case of $M = 1$ is straightforward. Consider $M \geq 2$. The tracker forms interior-cluster-disjoint 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 $\frac{M-1}{M}\underline{X}$.

B. Throughput Analysis

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

$$r_{CT}(M) = \frac{M-1}{M}\underline{X}. \quad (15)$$

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

$$P(\epsilon, M) = P\left(\bar{r}_{CT}(M) < (1 - \epsilon)\frac{M-1}{M}\mu\right) = P(\underline{X} < (1 - \epsilon)\mu). \quad (16)$$

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. 4 shows the \underline{n} versus N relationship in (17) with parameters that satisfy the condition in (18).

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

$$\underline{n} = \left\lceil \frac{2(1 + \alpha)\sigma^2}{\epsilon^2\mu^2} \ln N \right\rceil \quad (17)$$

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

$$P(\epsilon, M) = P(\min_{1 \leq i \leq K} X_i < (1 - \epsilon)\mu) < \frac{1}{2 + 2\alpha} \left(\frac{\epsilon\mu}{\sigma}\right)^2 \frac{1}{N^\alpha \ln N}, \quad (18)$$

¹For 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 $3M$.

which diminishes to zero as N goes to infinity.

Proof: Please refer to Appendix B. ■

Remarks: 1) Let $M = O(\ln N)$, then the minimum supported rate $r_{CT}(M) \geq (1 - \epsilon)\mu$ with high probability. 2) As long as every cluster is large enough and thus 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. Moreover, it indicates that high streaming rate can be obtained for finite N as long as we maintain large cluster sizes. We discuss the finite N case in Section IV-D and V-B. 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. 4) Bubble algorithm does not handle helper nodes, and our results for out-degree bounded problem are for help free scenario. However, Cluster-Tree algorithm can handle helper nodes because it is built from Mutualcast clusters which can handle helper nodes.

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 then 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 $X_i = \mu, \forall 1 \leq i \leq K$? 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

$$\underline{n} + 1 + 3M = \left\lceil \frac{2(1 + \alpha)\sigma^2}{\epsilon^2\mu^2} \ln N \right\rceil + 1 + 3M. \quad (19)$$

We next bound delay. There are a few flavors of delay in P2P streaming, and there have been several works studying the bound of delay [11], [25], [26]. 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 [5] 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 by²

$$\left(3z_d + (2\underline{n} + 2 + M) \frac{L}{\min_{v \in V} C_v} \right) \log_M \lceil N / \underline{n} \rceil. \quad (20)$$

Combining the above playback delay bound and Theorem 2, we observe that varying the required cluster size \underline{n} in Cluster-Tree algorithm can result in trading streaming rate with playback delay. Given N , M and α , large \underline{n} leads to small ϵ according to (17) and hence high $(1 - \epsilon)\mu$. However, according to (20), 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, [26] derived a bound on minimum playback delay that is smaller than the bound in (20). However, the streaming rate achievable under the minimum playback delay above is significantly lower than the streaming capacity achievable through the Cluster-Tree algorithm.

D. Bootstrap and 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 [27], 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 (18). Finally, the tracker computes the minimum required cluster size \underline{n} for a system with \bar{N} nodes according to (17). 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 $2\underline{n}$, the tracker splits the cluster into two, each with size \underline{n} . As more nodes join, they get added into existing clusters, and clusters grow and split whenever critical size $2\underline{n}$ 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 $2\underline{n} + 1 + 3M$.

With peer churn, the cluster size may decrease below \underline{n} . The tracker should make sure the minimum cluster size remains larger than \underline{n} . Whenever a peer joins, the tracker assigns it to the minimum-size cluster at that time. If the size of one of several cluster goes below \underline{n} , the tracker can merge them and other bigger clusters until the merged cluster has size well

above \underline{n} , for example $2\underline{n}$. There could also be other ways to actively prevent cluster sizes to go below the critical value \underline{n} .

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 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. According to the discussion in Section III-C, the M -ary Bubble trees are essentially CoopNet/SplitStream trees [6], [7]. Thus the associated overhead of maintaining the M -ary Bubble trees is the same as CoopNet/SplitStream trees.

V. NUMERICAL EVALUATION ON P2P TOPOLOGIES

The algorithms developed in this paper can be embodied into the control/management plane of P2P streaming systems. In our simulations, we choose the node uplink capacities randomly from a distribution that is obtained from real peer usage data as reported in [27]. The possible uplink capacities of peers and their respective fractions in the peer population are summarized in Table III. 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 III
PEER UPLINK CAPACITY DISTRIBUTION

A. Bubble Algorithm

We first illustrate the streaming rate r versus service capacity C_s curves for Bubble algorithm for different M values (throughout the simulations, we use M to denote both node degree bound and out-degree bound), and they are demonstrated in the left plot of Fig. 6, which shows that the Bubble rate curve is higher as M becomes larger, and it is very close to that of Mutualcast (unbounded streaming capacity) if $M \geq 6$. The important observation is that, although Bubble guarantees a 1/2 optimality for the worst case, it can actually achieves much better than 50% and very close to the exact optimality for practical peer bandwidth distributions.

We next compare the streaming rate achieved by several algorithms: Bubble (node out-degree bounded, 1/2 guarantee of streaming capacity), Snowball [3], [4] (achieving per-tree out-degree bound streaming capacity), CoopNet/SplitStream [6], [7] (node degree bounded, but no performance guarantee). We normalize the streaming rates achieved by these algorithms by the unbounded streaming capacity, which is achieved by Mutualcast algorithm [1]. We plot the normalized rates versus source uplink C_s for $M = 6$ in the middle plot of Fig. 6, which shows that: 1) for the practical peer uplink distribution, the per-tree degree bound $M = 6$ does not affect the streaming capacity and Snowball always gives the same rate as Mutualcast; 2) Bubble is within 3% of optimality and does better than CoopNet by about 5-10%.

From the middle plot of Fig. 6, CoopNet, a practical node degree bounded algorithm also achieves a high streaming rate.

²Here, $3z_d$ 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 (20) 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 (20), especially if one head node is enough for many clusters.

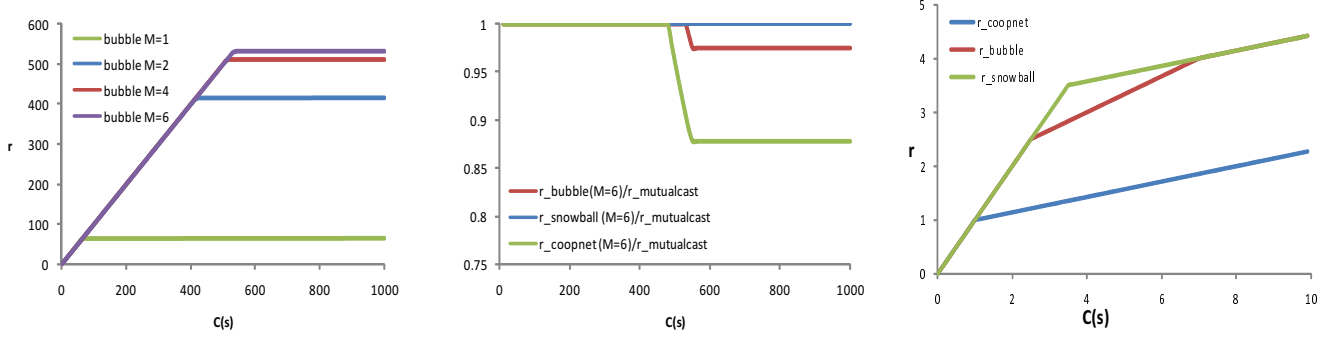


Fig. 6. Simulations on Bubble algorithm. Left: streaming rate versus source uplink C_s for Bubble algorithm, $N = 100,000$, $M = 1, 2, 4, 6$. It shows that the Bubble rate increases as M becomes higher, and the rate gain is marginal when $M \geq 6$. Middle, streaming rate ratio (w.r.t. Mutualcast) vs. source uplink C_s for Bubble, Snowball and CoopNet algorithms. It shows that when $M = 6$, Bubble rate is actually very close to the unconstrained streaming capacity, and is higher than CoopNet. Right: streaming rate versus source uplink C_s curve for Bubble, CoopNet, Mutualcast algorithms, for the special case of 7 peers. It shows that Bubble has performance guarantee while CoopNet does not, and for some cases, Bubble rate could be much larger than CoopNet rate.

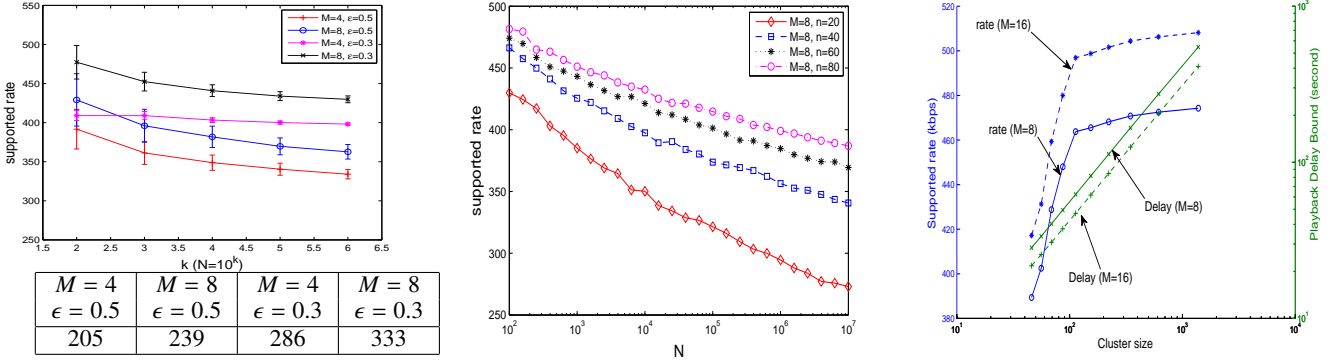


Fig. 7. Simulations on the Cluster-Tree algorithm. Left: Supported rates of the Cluster-Tree scheme vs. the number of peers N , for different M and ϵ . The table below the plot gives the corresponding theoretically guaranteed rate from Theorem 2. It shows that the actual supported rate for practical P2P systems could be much higher than the theoretically guaranteed rate. Middle: The same as Left curve, except that we fix the cluster size here to be 20, 40, 60, or 80. It shows that the supported rate is very good even if the cluster size is fixed, as long as this fixed size is not too small. Right: Supported rate (Kbps) vs. streaming delay (sec) of the Cluster-Tree scheme, as a function of the cluster size. It shows the trade-off between throughput and delay.

However, CoopNet does not provide any guarantee of rate sub-optimality. For example, consider the simplest topology of CoopNet with 7 nodes (say, A to G), as shown in Fig. 1 of [6]. For this simplest network, CoopNet construct two trees, where nodes A to C are internal in tree 1, and nodes D to F are internal in tree 2. Suppose the peer bandwidths are very heterogeneous, for example, $C = [10, 10, 1, 1, 1, 1]$. For that case, the rate achieved by CoopNet could be far from optimality, while Bubble still achieves a rate curve close to optimality, as demonstrated in the right plot of Fig. 6.

B. Cluster-Tree Algorithm

We first simulate on supported rate and validate Theorem 2. 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 III. 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 (17) and (18).

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 III. 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 the left plot of Fig. 7. We observe that actual rate achieved by Cluster-Tree algorithm for finite N is significantly higher than the asymptotic bound guaranteed by Theorem 2; the latter is shown in the table below the left plot of Fig. 7.

Two additional observations can be drawn from the left plot of Fig. 7. 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} . The second observation is that for a fixed N and M , small ϵ leads to small 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} .

We next study the supported rate if the cluster size has

to be fixed. Ideally Cluster-Tree scheme sets cluster size as logarithmic function of system size. However, there might be an upper bound on cluster size due to overhead concerns. If N exceeds our maximal supported system size, then the cluster size freezes at the upper bound. The performance with fixed cluster size is plotted in the middle plot of Fig. 7. It shows that the supported rate does not decay sharply as N increases, if the fixed size is not too small. This indicates that the Cluster-Tree scheme is robust against the system size and cluster size.

We finally study throughput and delay trade-off in the Cluster-Tree algorithm for $N = 10^6$, by varying the cluster size. According to (17), for fixed N , we can vary the cluster size by changing ϵ . For fixed M , the playback delay, according to (20), 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 the right plot of Fig. 7, we plot the throughput and delay as functions of cluster size n , for $M = 8$ and $M = 16$. It shows that, 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.

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 a rate close to the maximum rate achieved under no degree bound constraint. 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.

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] 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.
- [3] 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.
- [4] S. Liu, M. Chiang, S. Sengupta, J. Li, and P. A. Chou, "Streaming capacity for p2p with degree bound," in *Proceedings of Allerton Conference*, Monticello, IL, September 2008.
- [5] 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.
- [6] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in cooperative environments," in *Proc. ACM SOSP*, New York, NY, October 2003.

- [7] V. Padmanabhan and K. Sripanidkulchai, "The case for cooperative networking," in *Proceedings of IPTPS*, Cambridge, MA, March 2002.
- [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] 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.
- [13] Y. Guo, C. Liang, and Y. Liu, "dHPCS: decentralized hierarchically clustered p2p video streaming," in *Proc. ACM International conference on Content-based Image and Video Retrieval*, New York, NY, July 2008.
- [14] F. López-Fuentes and E. Steinbach, "Hierarchical collaborative multicast," in *Proc. the 15th International Conference on Multimedia*, Augsburg, Germany, September 2007.
- [15] S. Shi and J. Turner, "Multicast routing and bandwidth dimensioning in overlay networks," *IEEE Journal on Selected Areas in communications*, vol. 20, no. 8, pp. 1444–1455, 2002.
- [16] Z. Li, B. Li, and L. Lau, "On achieving maximum multicast throughput in undirected networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, p. 2485, 2006.
- [17] Y. Zhu and B. Li, "Overlay networks with linear capacity constraints," *IEEE Transactions on Parallel and Distributed systems*, vol. 19, no. 2, pp. 159–173, 2008.
- [18] C. Feng, B. Li, and B. Li, "Understanding the performance gap between pull-based mesh streaming protocols and fundamental limits," in *Proc. of IEEE INFOCOM*, 2009.
- [19] R. L. C. L. Abeni, C. Kiraly, "On the optimal scheduling of streaming applications in unstructured meshes," in *IFIP Networking*, 2009.
- [20] 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*, vol. 25, no. 9, p. 1655, 2007.
- [21] X. Zhang, J. Liu, B. Li, and T. Yum, "CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming," in *proceedings of IEEE Infocom*, vol. 3, 2005, pp. 13–17.
- [22] D. A. Tran, K. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *In Proc. of IEEE Infocom*, 2003.
- [23] "PRIME: Peer-to-peer receiver-driven mesh-based streaming," in *In Proc. of IEEE Infocom*, 2007.
- [24] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, 1979.
- [25] 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.
- [26] Y. Liu, "Delay bounds of peer-to-peer video streaming," nYU/Polytechnic Institute Technical Report, Jun. 21, 2009.
- [27] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" in *Proc. ACM SIGCOMM*, Kyoto, Japan, August 2007.

APPENDIX

A. Proof of Results in Section III

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 (10). 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 4, and the rigorous proof is available in a longer version of this paper. 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 now prove Lemma 3.

Proof: We first prove 1). For algorithm A, if node j is left unexhausted, then $Mr < C_j$. For all $i < j$, $C_i \geq C_j > Mr$, 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 $r_B^* > r_A^*$. Meanwhile, from [3], [4], we have:

$$r_A^* = \frac{\sum_{k>j_A} C_k}{N-1-Mj_A} \quad (21)$$

$$r_B^* = \frac{\sum_{k>j_B} C_k}{N-1-Mj_B} \quad (22)$$

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,

$$\begin{aligned} r_B^* &= \frac{1}{N-1-Mj_B} (C_{j_B+1} + \dots + C_{j_A} + \sum_{k>j_A} C_k) \\ &\leq \frac{1}{N-1-Mj_B} (Mr_B^* + \dots + Mr_B^* + \sum_{k>j_A} C_k) \\ &= \frac{1}{N-1-Mj_B-M(j_B-j_A)} \sum_{k>j_A} C_k \\ &= \frac{1}{N-1-Mj_A} \sum_{k>j_A} C_k = r_A^* \end{aligned} \quad (23)$$

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 (21), $r_A^* = r_B^*$ 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 consider 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 the case of $N \neq 1 + IM$, the proof is similar but a little bit more complicated, and it is available in a longer version of this paper.

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 -th 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 $\{1, 2, \dots, N\}$ to three sets:

$$\begin{cases} A &:= [1, J_1] \\ B &:= [J_1 + 1, I] \\ C &:= \{i : I < i \leq N, \tilde{C}_i > 0\} \\ D &:= \{i : I < i \leq N, \tilde{C}_i = 0\} \end{cases} \quad (24)$$

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

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

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

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

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

$$r_b^* \geq \frac{\sum_{v \in B} C_v}{N-1-J_1} > \frac{\sum_{v \in C} C_v}{N-1-J_1} \geq \frac{I}{I-1} r_e \quad (25)$$

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

From Figure 4 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 (25), we have proved (11). ■

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

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

We finally prove Proposition 3.

Proof: For the case of homogeneous receivers, interior nodes of all Bubble trees are disjoint. 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 $\lfloor N / (\lceil \frac{N-1}{M} \rceil) \rfloor$.

For $N > M+1$, there exist $k \geq 1$ 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,

$$\left\lfloor \frac{N}{k+1} \right\rfloor \leq \left\lfloor \frac{N}{\lceil \frac{N-1}{M} \rceil} \right\rfloor \leq \left\lfloor \frac{N}{k} \right\rfloor. \quad (26)$$

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 (26), 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 $\frac{M-1}{M}C$. ■

B. Proof of Results in Section IV

The proof of Proposition 4 is straightforward. The total rate needed is $X_i n_i$ and the total available capacity is also $X_i n_i$. If we get additional supply from outside at X_i , then we have X_i extra to serve outside.

We now 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 these nodes be v_1 . If $C_{v_1} \geq 2X_i$, then we are done. If $2X_i > C_{v_1} \geq X_i$, 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: By union bound, we have

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

If $\underline{n} = \lceil \frac{2(1+\alpha)\sigma^2}{\epsilon^2\mu^2} \ln N \rceil$, 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 [24, Theorem 4.3].

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

Consider a corresponding average-bin-weight instance with $n = 4|\mathcal{M}|$ 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 $p[1], p[2], \dots, p[N(p)]$, where $N(p)$ denotes the number of triples from \mathcal{M} in which p occurs. These balls' weights are:

$$\begin{aligned} w(x_i[l]) &= \begin{cases} 10(32q)^4 + i \cdot 32q + 1, & 1 \leq i \leq q, l = 1; \\ 11(32q)^4 + i \cdot 32q + 1, & 1 \leq i \leq q, 2 \leq l \leq N(x_i). \end{cases} \\ w(y_j[l]) &= \begin{cases} 10(32q)^4 + j(32q)^2 + 2, & 1 \leq j \leq q, l = 1; \\ 11(32q)^4 + j(32q)^2 + 2, & 1 \leq j \leq q, 2 \leq l \leq N(y_j). \end{cases} \\ w(z_k[l]) &= \begin{cases} 10(32q)^4 + k(32q)^3 + 4, & 1 \leq k \leq q, l = 1; \\ 8(32q)^4 + k(32q)^3 + 4, & 1 \leq k \leq q, 2 \leq l \leq N(z_k). \end{cases} \end{aligned}$$

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 we show the following problems are NP-Complete.

- **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, by [24, Theorem 4.3], the desired 4-partition exists if and only if \mathcal{M} contains a matching, i.e., this instance of 4-PARTITION is as hard as 3DM and is NP-Complete.

On the other hand, 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, 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 $B/4$.

Hence this instance of average-bin-weight problem is also as hard as 3DM and is NP-Complete. ■