P2P Streaming Capacity: Survey and Recent Results

Minghua Chen, Mung Chiang, Phil Chou, Jin Li, Shao Liu, and Sudipta Sengupta

***Abstract*— Peer-to-peer (P2P) systems provide a scalable way to stream content to multiple receivers over the Internet and has become a major type of application traffic. The maximum rate achievable by all receivers is the capacity of a P2P streaming session. We provide a taxonomy of the problem formulations. In each formulation, computing P2P streaming capacity requires the computation of an optimal set of multicast trees, generally with an exponential complexity. We survey the family of constructive, polynomial-time algorithms that can compute P2P streaming capacity and the associated multicast trees, arbitrarily accurately for some of the formulations, and to some approximation factors in other formulations. Performance evaluation using large-scale Internet trace is provided before open problems in this research area are discussed.**

I. Background and Modeling

To deliver rich multimedia content to many users, a popular recent trend is to employ a peer assisted network to complement the existing data center and CDN infrastructure. With all the promising potential of P2P content distribution, especially streaming video, we are naturally led to the following fundamental question: What is the limit of such networks? And how can we design P2P algorithms to attain the limit? Defining *streaming capacity* as the maximum supported streaming rate that can be received by every receiver, these two questions, of determining the streaming capacity of P2P, and of achieving (exactly or arbitrarily closely) the capacity, are being addressed by the research community. This paper provides a survey of these results, with more detailed discussions of the recent results of Bubble algorithm, Cluter-tree algorithm, and performance evaluation using large scale real traffic traces.

A. P2P Network Modeling

We use a directed graph, denoted by $G = (V, E)$, to model a P2P overlay network. A vertex v in V corresponds to a node on the P2P overlay networks, and every edge e in E corresponds to a TCP/UDP connection between two endpoints. We model P2P networks as networks with capacity constraints on nodes, rather than on links.

In practical P2P networks, there are many factors that limit system throughput. For example, a peer node has limited download capacity to receive its own data, and limited upload capacity to forward content to other peers. Many peers are simply commercial home PCs with limited memory and/or CPU power, and can only maintain connections to a small subset of peers. This constraint limits the number of other peers a receiver node can help, and may reduce system throughput. We define the *degree of a node* as the number of neighbors of the node, and thus as the number of simultaneous connections this node needs to maintain.

To model the node capacity constraints, we assume every edge e in E has infinite capacity and associate with each node v an upload capacity constraint $C_{out}(v)$, and a download capacity constraint $C_{in}(v)$. Let $In(v)$ and $Out(v)$ denote the set of edges entering and leaving node v respectively. Let $f(e) : e \to [0, \infty)$ be the rate over edge e in E. For each node v, we constrain the sum of the edge rate entering and leaving v as follows:

$$\sum_{e \in In(v)} f(e) \le C_{in}(v), \qquad \sum_{e \in Out(v)} f(e) \le C_{out}(v). \quad (1)$$

Let $S \subseteq V$ be a designated set of senders, and for each $s \in S$, let $R_s \subseteq V - \{s\}$ be a designated set of receivers. An information source originates from each sender $s \in S$, and is to be received by each $r \in R_s$. The remaining nodes $H_s = V - R_s - \{s\}$ are a set of helper nodes that can act as relays for distributing the information from source s.

There are two paradigms to multicast messages from a single source s to its set of receivers R_s. The first one is routing, i.e., each source s packs multiple directed Spanning/Steiner trees rooted at s and reaching all receivers in R_s. Every tree is responsible for delivering a subset of the broadcast messages, and the nodes only perform the actions of store, copy, and forward. The second one is network coding [1][2], and in particular random linear network coding [3]. In this paradigm, the source s starts by transmitting the message to its neighboring nodes. As the messages propagate through the network, each node generates coded messages as random linear combinations of its received messages and output the coded messages to its neighbors. Receivers obtain random linear combinations of the messages. They can recover the original batch of messages from any set of random linear combinations that form a full rank matrix.

Network coding is not quite practical in today's P2P applications. It cannot be used in the Internet routing layer because it requires changes in all routers (for encoding) and end-hosts (for decoding). If deployed in the overlay (P2P layer), it will introduce new complexity in end-host software (for encoding and decoding) and additional delays in video delivery. Consequently, almost all comercial P2P systems adopt the routing solution.

In the rest of this paper we focus on the following settings unless mentioned otherwise:

- single source multicast scenario over the P2P networks,
- with only node upload capacity constraints,
- and no network coding is allowed.

Authors appear in alphabetical order. Chen is affiliated with The Chinese University of Hong Kong, Chiang is affiliated with Princeton University, and the other authors with Microsoft.

II. Existing Work

We review existing work on maximizing P2P streaming rate in this section, and discuss insights behind these approaches. We start from the simplest scenarios and proceed by introducing practical constraints.

A. Full-mesh Networks, No Degree Bound

Streaming capacity over a full-mesh P2P network without node degree bound is a well studied problem. Li et al. [4] have defined the capacity of a full-mesh P2P network with helpers, and shown that the capacity can be achieved through so called MutualCast trees. Kumar et al. [5] and Massoulie et. al. [6] have independently developed the same result for the maximum broadcasting rate in a full-mesh heterogenous P2P network, but without helpers. Nguyen et al. [7] have studied the throughput efficiency of the full-mesh P2P network, and arrived at the same result for homogeneous P2P networks without helpers.

Specifically, for single source multicast in P2P networks with only upload capacity constraints and no node degree bound, the following results hold [4]:

Theorem 1: Consider the case where source s multicasts to all its receivers with helper nodes in presence. The maximum streaming rate can be achieved with design complexity that is polynomial-time in network size by packing at most $1 + |R_s| + |H_s|$ multicast trees as follows:

- One depth-1 tree rooted at s and reaching all receivers in R_s, i.e. the type (1) tree in Fig 1.
- $|R_s|$ depth-2 trees, each rooted at s and reaching all other receivers in R_s via different $r \in R_s$, i.e. the type (2) tree in Fig 1.
- $|H_s|$ depth-2 trees, each rooted at s and reaching all receivers in R_s via different $h \in H_s$, i.e. the type (3) tree in Fig 1.

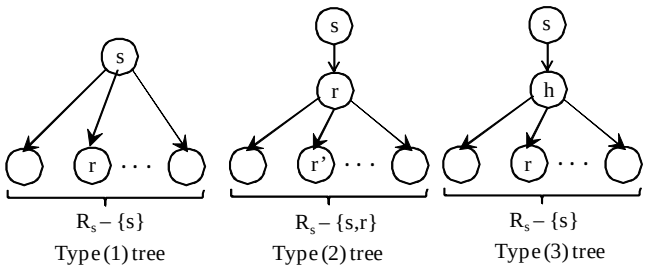

Fig. 1. Packing different types of multicast trees to achieve maximum streaming rate.

This result extends Edmonds' theorem [8] for P2P topology.

Intuitive explanation: The streaming rate is determined by a) the system utilization of total system capacity, and b) the transmission efficiency. To maximize the streaming rate, it is sufficient to maximize the system utilization and the transmission efficiency. The first, second, and third type of trees in Fig 1 maximize the capacity utilization of the sender, individual receivers, and individual helpers, respectively. Meanwhile,the first two types of trees deliver one bit of information to $|R_s|$ receivers by using exactly $|R_s|$ bits of system capacity, resulting in full transmission efficiency. The third type of trees deliver one bit of information to $|R_s|$ receivers using $|R_s|+1$ bits of system capacity, where the one extract bit consumption is due to a helper must receive the information before rebroadcasting to the receivers. This leads to a transmission efficiency of $\frac{|R_s|}{|R_s|+1}$. This is the maximum possible value for transmission that involves helpers, since helpers need to receive at least one copy of the message before they can help in redistributing the message. Thus, by properly packing the three types of trees in Fig 1, one can maximize the system utilization and also the transmission efficiency, and hence the streaming rate.

s, V, R	the source, set of all nodes, set of all receivers.
C_v	upload capacity of node $v \in V$.
$U(v)$:	Actual upload of node $v \in V$.
$\mathbb{T}$:	Set of all feasible trees.
t	a tree, $t \in \mathbb{T}$.
T	a multi-tree, a set of sub-trees.
y_t	rate of substream delivered by tree t.
$m_{v,t}$	out-degree of node v in tree t.
M	uniform per-tree out-degree bound (**p.t.d.b.**).
$M(v)$	non-uniform 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.**).
$\bar{r}_N(C_s, D_o)$	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 uniform per-tree out-degree bound M.

TABLE I

Main notation used in this paper.

However, the tree-packing scheme of Theorem 1 cannot be directly applied to practical P2P networks. This is because the scheme requires each peer to connect to each of the remaining $|R_s| - 1$ receiver peers simultaneously. Thus the required node degree is unbounded as the network size increases, whereas in realistic networks the node degree is bounded to limit the overhead in maintaining simultaneous connections to neighboring nodes. For instance, in practical systems such as PPLive [9], the total number of neighbors of a node is usually bounded around 200 [10], and the number of active neighbors of a node is usually bounded by $10-15$.

B. Fullmesh Network, Bounded Per-Tree Degree

The first step toward a constrained tree construction is to limit the per-tree degree bound. The streaming capacity problem under this constraint is formulated as below:

Streaming Capacity Problem

$$\text{maximize} \quad r = \textstyle\sum_{t \in T} y_t \tag{2}$$

$$\text{subject to} \quad \textstyle\sum_{t \in T} m_{v,t} y_t \le C_v\,, \quad \forall\, v \in V \tag{3}$$

$$y_t \ge 0, \quad \forall\, t \in T \tag{4}$$

$$m_{v,t} \le M(v), \quad \forall\, t \in T, v \in V \tag{5}$$

$$\text{variables} \qquad T \subset \mathbb{T}, \text{ and } y_t, \forall t \in T\,. \tag{6}$$

With this constraint, to support a streaming rate r, any receiver v can at most upload $D(v,r) := \min(M(v)r, C_v)$, called effective uplink capacity. With the new maximum upload, the new upper bound for streaming capacity is given in [11]:

$$\begin{aligned} & r_{max}(C_s, C_v, M(v)) \\ \le & \min\left(C_s, \tfrac{1}{N}\left(C_s + \textstyle\sum_{v=1}^{N} D(v,r)\right)\right) \\ = & \min\left(C_s, \tfrac{1}{N}\left(C_s + \textstyle\sum_{v=1}^{N} \min(M(v)r, C_v)\right)\right)\,. \end{aligned} \tag{7}$$

Since r appears in both side of the inequality, it is easier to solve an equivalent problem, called minimum server load problem, which seeks the minimum server upload $U_{min}(s)$ that is sufficient to support a given streaming rate r. Minimum server load and streaming capacity are equivalent and once we solve one, we can solve the other:

$$\begin{aligned} & r_{max}(C_s, C_v, M(v)) \\ = & \max\left(r \in R^+ : U_{s,min}(r, C_v, M(v)) \le C_s\right), \quad (8) \\ & U_{s,min}(r, C_v, M(v)) \\ = & \min\left(C_s \in R^+ : r_{max}(C_s, C_v, M(v)) \ge r\right). \quad (9) \end{aligned}$$

For the server load, we have the following lower bound:

$$U_{min,s}(r, C_v, M(v)) \ge \max\left(r, Nr - \sum_{v=1}^{N} \min(M(v)r, C_v)\right) \tag{10}$$

It is shown in [11], [12] that the bounds in (7) and (10) are indeed achievable. The proof is by constructing a "Bottleneck Removal algorithm" for $M \equiv 1$, a "Rotation algorithm" for uniform $M > 1$ and homogeneous peer capacities, and a "Snowball algorithm" for uniform $M > 1$ and heterogeneous peer capacities, and a variant of Snowball algorithm for the most general case, non-uniform $M(v)$ and heterogeneous peer capacities. It is shown that the four algorithms achieve the bounds for the four respective cases.

Combining all cases, we get the following result.

Theorem 2: The four algorithms mentioned above can achieve both the lower bound for server load and the upper bound for streaming rate for the four respective cases, and thus solve the minimum server load problem and the streaming capacity problem under per-tree node degree bound. Furthermore,

1) For minimum server load, we have

$$U_{s,min}(r) = \max(r, Nr - \sum_{v \in V} \min(C_v, M(v)r))\,, \tag{11}$$

2) For the streaming capacity, we have

$$r_{max}(C_s, C_v) = \begin{cases} C_s & \text{if } C_s \le P(N)\,, \\ \min(C_s, g(k^*)) & \text{if } C_s > P(N)\,, \end{cases} \tag{12}$$

where

$$g(k) := \begin{cases} \frac{C_s + \sum_{i=k}^{N} C_i}{N - \sum_{i=1}^{k-1} M(i)} & \text{if } \sum_{i=1}^{k-1} M(k) < N\,, \\ \infty & \text{otherwise,} \end{cases} \tag{13}$$

and k^* is the minimum k such that

$$P(k) \le g(k, M) \quad \text{and} \quad P(k) \le C_s\,. \tag{14}$$

3) The degree bound is *inactive*, i.e., the streaming capacity and minimum server load with degree bound always equal to those without degree bound, if and only if

$$\frac{\sum_{v=2}^{N} C_v}{N - M(1) - 1} \ge P(1)\,. \tag{15}$$

Otherwise, the degree bound is *active*, i.e., there exists a C_s value, such that $r_{max}(C_s, M) < r_{max}(C_s, \infty)$, or there exists a r value, such that $U_{s,min}(r, M) > U_{s,min}(r, \infty)$.

Theorem 2 shows the optimality of the four algorithms, derives the exact formula for streaming capacity and minimum server load, and gives the necessary and sufficient condition on the activeness of the degree bound. From the condition, if the peer capacities are close to each other, the degree bound is inactive; if the peer capacities differ too much, the degree bound may be active.

C. Fullmesh Network, Bounded Node Degree

Bounding the total number of neighbors for every node can reduce the achievable streaming capacity. For instance, if each node can have at most two neighbors, then the only possible streaming topology is a chain. Consequently, the maximum streaming rate is bounded by the second to the minimum node capacity, which can be significantly worse than the maximum streaming rate without degree bound.

What the server can do is to optimize the neighbor lists it gives to each peer in order to improve the streaming rate. To proceed, we first consider the scenario where no helper nodes are present, all nodes have the same upload capacity, i.e., $C_v = C, \forall v \in V$, and each node has a same node degree bound D. For the sake of easy explanation, we assume D to be an even number. The analysis for D being an odd number can be carried out similarly.

For this homogeneous broadcasting scenario, CoopNet [13] and SplitStream [14] constructs multiple interior-node-disjoint trees with tree degree $\frac{1}{2}D$ to perform the streaming. For instance, consider an example of the server streams to seven nodes all having unit upload capacity, and the node degree bound is 4. The CoopNet and SplitStream schemes construct two trees with tree degree 2 as shown in Fig. 2. Each tree delivers half of the video at rate $\frac{1}{2}C$, and hence the total achieved streaming rate is the optimal value C. The interior nodes on these two trees are disjoint. this is because all interior nodes use up their capacities simultaneously on one tree, and are leaf nodes in the other tree.

The above observations on CoopNet/SplitStream schemes can be generalized. The following lemma shows that CoopNet [13] and SplitStream [14] can achieve a streaming rate of at least $\frac{D-2}{D}C$, which is close to the maximum possible rate C for large D.

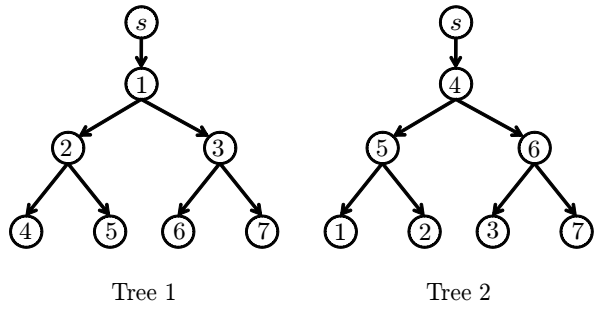

Fig. 2. In this example, the server s forms two interior-node-disjoint trees, each having tree degree 2, to stream video to all seven receivers. In tree 1, nodes $1, 2, 3$ are interior nodes; while in tree 2, they become leaf nodes and nodes $4, 5, 6$ become the interior nodes. There are not enough nodes with non-zero upload capacity left to build a third tree; hence, node 7 does not get the chance to serve as an interior node.

Lemma 1: [15]Consider that CoopNet/SplitStream schemes construct multiple degree-$D/2$ trees in a single-source streaming scenario with N homogeneous receivers, i.e., $C_v = C$ for all v in R, and $N > \frac{D}{2} + 1$. The followings are true:

- The total number of CoopNet/SplitStream trees is either $\frac{D}{2} - 1$ or $\frac{D}{2}$, and the trees are interior-node-disjoint;
- Node degree is bounded by D;
- Streaming rate supported by these trees is at least $\left(1 - \frac{2}{D}\right) C$.

Intuitive explanation: To maximize the streaming rate from the server to N homogeneous receivers with degree bound D, it is again critical to fully utilize receivers' upload capacities while maintaining high transmission efficiency. One straightforward way is to construct trees for delivery and let the receivers to serve as interior nodes. If all interior nodes of a tree have the same number $\frac{D}{2}$ of children, then the all interior nodes use up their capacities simultaneously on one tree and should be leaf nodes in the rest trees. Hence, the trees are interior-node-disjoint. This turns out to be key to fully utilize receivers' capacities while bounding the node degree. A tree at most uses up $\left\lceil \frac{N}{D/2} \right\rceil$ number of interior nodes' capacities, and is capable of support a rate of $\frac{C}{D/2}$. The the total number of CoopNet/SplitStream trees is either $\frac{D}{2} - 1$ or $\frac{D}{2}$. Consequently, the achieved streaming rate is at least $\left(\frac{D}{2} - 1\right) \frac{C}{D/2} = \left(1 - \frac{2}{D}\right) C$. Across multiple trees, a node in total has at most $\frac{D}{2}$ children and $\frac{D}{2}$ parents; hence, its node degree is bounded by D.

Now consider the case where N receivers have heterogenous upload capacities. Let $\mu \triangleq \frac{1}{N} \sum_{v \in R} C_v$ be the average receiver upload capacity. Assuming (as is usually true) that the server upload capacity is larger than μ, for a sufficiently large number of peers it can be verified by a supply-equals-demand argument that μ is approximately the maximum streaming rate.

In this case, directly applying CoopNet/SplitStream schemes can no longer guarantee the interior nodes of a tree use up their capacities simultaneously. Without the interior-node-disjoint property, it becomes complicated to fully utilize receivers' capacities while bounding the node degree across multiple trees.

1) Bubble Algorithm: We then extend our study of the streaming capacity to node out-degree bound (**n.o.d.b.**), i.e., $M(v) = D_o < \infty$. It is shown in [15] that the streaming capacity problem under node degree bound on top of a general graph is NP-Complete in general. It remains unknown whether the streaming capacity problem under n.o.d.b. on top of a complete graph is NP-hard or not, though. We also derive the upper and lower bounds of the n.o.d.b. streaming capacity, denoted by $\bar{r}_N(C_s, D_o)$, in [15].

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

$$\bar{r}_N(C_s, D_o) \leq \bar{r}_T(C_s, M)|_{M=D_o}. \tag{16}$$

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

Theorem 3: 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.,

$$\bar{r}_N(C_s, D_o) \geq \frac{1}{2} \bar{r}_T(C_s, M)|_{M=D_o}, \forall C_s \geq 0, \forall D_o, N \geq 1. \tag{17}$$

We prove Theorem 3 by constructing a "Bubble algorithm" that satisfies n.o.d.b., and achieves at least half of the Snowball capacity. 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". By 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 not a practical P2P streaming scheme. We design it to prove Theorem 3. It is also the first algorithm that satisfies n.o.d.b. and has provable performance guarantee.

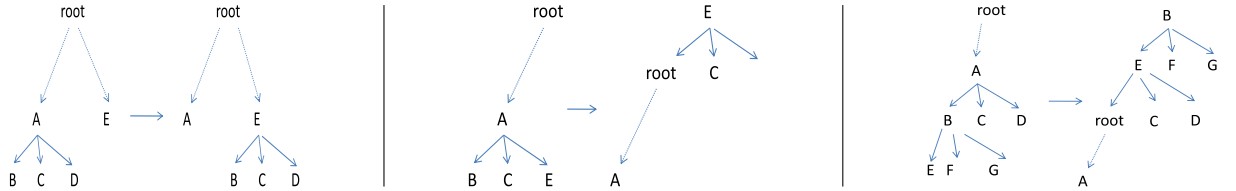

Fig. 3. The swap of an exhausted internal node and a leaf node. From left to right, case 1, case 2 and case 3. Case 1: A is not an ancestor of E. Case 2: A is the parent of E. Case 3: A is an ancestor, but not the parent, of E.

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. 4. 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).

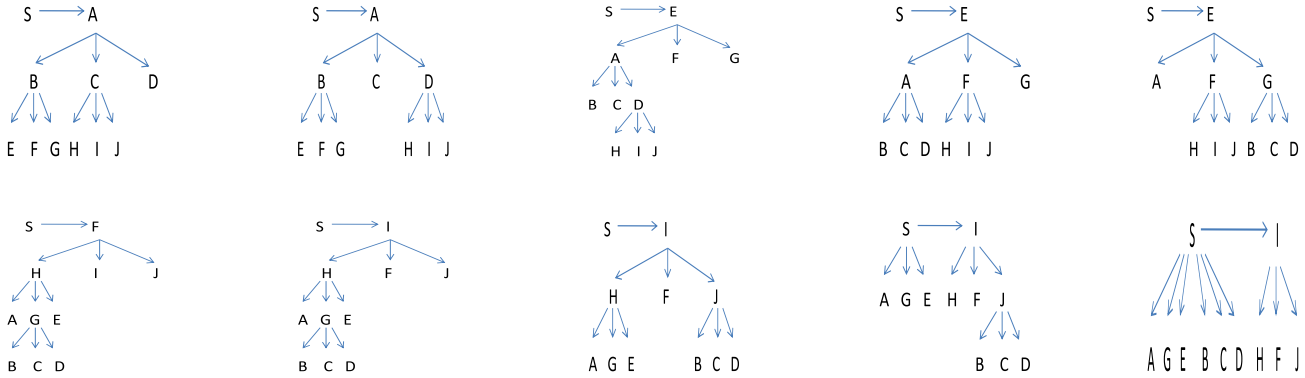

Fig. 5. Constructed trees in all iterations for the example in Fig. 4. The first 8 trees are initial trees. After iteration 8, there are less than $I = 3$ peers left unexhausted, and Bubble algorithms stops. Then server takes care of the children of all exhausted internal nodes from then on.

The key challenge of Bubble algorithm 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. 3.

- **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.

A complete example of how the Bubble algorithm constructs the multi-tree to utilize peer bandwidths is shown in Fig. 4 and Fig. 5. 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. 4, and the trees constructed in the iterations are shown in Fig. 3.

With the Bubble algorithm at hands, we have furthermore shown the following result, which directly leads to Theo-

rem 3.

Proposition 1: 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

$$\frac{r_b(C_s, D_o)}{r_s(C_s, M)} \geq \frac{I}{2I-1} + \frac{I-1}{(2I-1)(1+M)}. \quad (18)$$

From Proposition 1, we have:

$$\bar{r}_N(C_s, D_o) \geq r_b(C_s, D_o) \geq \frac{1}{2}\bar{r}_T(C_s, M)|_{M=D_o} \quad (19)$$

$$r_b(C_s, D_o) \geq \frac{1}{2}\bar{r}_T(C_s, M)|_{M=D_o} \geq \frac{1}{2}\bar{r}_N(C_s, D_o) \quad (20)$$

Equation (19) states that the n.o.d.b. streaming capacity is at least half of the p.t.d.b. streaming capacity, which is Theorem 3. Equation (20) states that Bubble algorithm guarantees a $1/2$ sub-optimality in term of rate performance.

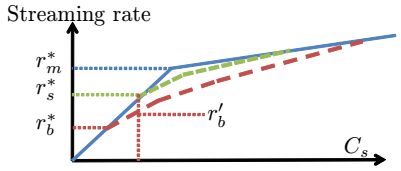

Fig. 6. The critical rates and streaming capacity functions of Bubble (bottom red curve), Snowball (middle green curve) and Mutualcast (top blue curve) algorithms. r_m^*, r_s^*, r_b^* are the corresponding critical rates. r_b' is the streaming rate of Bubble algorithm when $C_s = r_s^*$. The worst $r_b(C_s, D_o)/r_s(C_s, M)|_{M=D_o}$ ratio occurs at $C_s = r_s^*$, and it is r_b'/r_b^*.

Proposition 1 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. 6.

2) Cluster-Tree Algorithm: A common message illustrated by CoopNet/SplitStream and Bubble schemes is that streaming to homogenous receives is straightforward, but to heterogenous is rather complicated. One way to proceed is to create homogeneity out of heterogeneity. Following this philosophy, we construct a two-layer hierarchical scheme to aim for a rate close to μ.

We assume that for all v in $V - \{s\}$, $C_{out}(v)$ is drawn i.i.d. from a distribution with mean μ and variance σ^2. Our idea is two folds. First, we group peers into clusters and form a full-mesh network to deliver content locally within a cluster. Second, we use CoopNet/SplitStream trees to deliver contents globally across groups. Locally, the rates supported by different clusters turns out to be the average node upload capacities in the clusters. These average node capacities are roughly the same if clusters are large enough, according to the Central Limit Theorem. Globally, using degree bounded CoopNet/SplitStream trees to deliver content across these equal-rate clusters achieves high throughput. An illustrative example consisting seven clusters is shown in Fig. 7. This example can be thought as an counterpart to the seven homogeneous receivers example related to CoopNet/SplitStream.

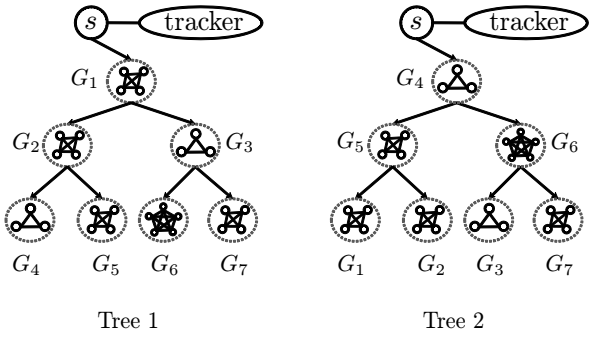

Fig. 7. In this example, the server groups the peers into seven clusters. The server s forms two interior-node-disjoint trees, each having tree degree 2, to distribute video to all clusters. Within each cluster, the peers form a full-mesh and locally broadcast the video among themselves. In tree 1, $G1, G2, G3$ are interior clusters; while in tree 2, they become leaf clusters and $G4, G5, G6$ become the interior clusters.

In particular, we distribute the N receivers *evenly* into K clusters $G_1, \ldots, G_K$; let X_i be the average peer upload capacity in G_i, i.e.,

$$X_i = \frac{1}{|G_i|} \sum_{v \in G_i} C_{out}(v). \quad (21)$$

We form a full-mesh among the peers inside clusters, and apply the scheme in Theorem 1 in Section I-A to deliver content within cluster $G_i (1 \leq i \leq K)$ at rate X_i. If G_i gets content from outside at rate X_i, then not only can all peers inside G_i receive this data, but also the cluster can act as a source and send a copy of the data to nodes outside G_i at rate X_i. By connecting all the clusters and the source via a chain, we see that a streaming rate of $\bar{X} \triangleq \min_{i=1,\ldots,K} X_i$ is achievable. If alternatively degree-M CoopNet/SplitStream trees are in use, then a rate of $\frac{M-1}{M}\bar{X}$ is achievable.

Observing that each X_i is the sum of i.i.d. random variables and thus follows a normal distribution concentrating around its mean μ, we know that the probability of the tail of the distribution decreases exponentially in the cluster size $\frac{N}{K}$. Therefore, if $K = O(\ln N)$, the probability $\bar{X}$ is far away from μ would be upper-bounded by $o\left(\frac{1}{\ln N}\right)$ using the union bound argument. In particular, the probability that $\bar{X}$ is smaller than $(1-\epsilon)\mu$ for $\epsilon \in (0, 1]$ goes to zero as N goes to infinity. That is, $\bar{X}$ also concentrates around μ if $K = O\left(\frac{N}{\ln N}\right)$. If the CoopNet/SplitStream trees in use have a degree no more than $O(\ln N)$, then the overall node degree is bounded by $O(\ln N)$. The above observation is formally summarized in the following theorem.

Theorem 4: If N peers, with independently identically distributed upload capacities, are evenly distributed into $K = \lceil N/\underline{n} \rceil$ clusters where

$$\underline{n} = \left\lceil \frac{2(1+\alpha)\sigma^2}{\epsilon^2 \mu^2} \ln N \right\rceil \quad (22)$$

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

$$P\left(\min_{1 \leq i \leq K} X_i < (1-\epsilon)\mu \right) < \frac{1}{2+2\alpha} \left(\frac{\epsilon\mu}{\sigma}\right)^2 \frac{1}{N^\alpha \ln N}, \quad (23)$$

which diminishes to zero as N goes to infinity.

Using degree-M CoopNet/SplitStream trees to deliver content across these K clusters, we achieve a rate of at least $\frac{M-1}{M}\mu$ with probability no less than $1-\frac{1}{2+2\alpha}\left(\frac{\epsilon\mu}{\sigma}\right)^2\frac{1}{N^\alpha \ln N}$. The degree of each node is $O(\ln N + M)$.

The insight is 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.

Based on the above insight, we develop a Cluster-Tree algorithm to achieve high streaming rate with bounded node degree. The Cluster-Tree algorithm is of a practical design: a distributed algorithm and it can handle peer churns.

We consider a P2P streaming system with tracker. Trackers have been adopted in many practical P2P streaming systems, such as PPLive [9] and PPStream [16], 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 [17], a small number of tracker servers are able to manage possibly millions of streaming users.

The Cluster-Tree algorithm works as fellow. 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 the average upload capacity μ and the variance of capacity σ^2 from a third-party statistics [18], 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 (23). Finally, the tracker computes the minimum required cluster size n^* for a system with $\bar{N}$ nodes according to (22). 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 [14], [19].

D. General Network, Bounded Tree/Node Degree

Given a general (i.e., non-full-mesh) P2P network, bounding the node degree significantly complicate the streaming capacity problem, making it combinatorial in nature and is computationally infeasible to solve in general. In particular, if the node degree is bounded by 2, then this is a Hamiltonian path problem and is NP-complete [20].

One way to proceed is to consider for a variant of the node degree bound problem. In our recent work [21], we consider the problem of packing multicast trees over a given P2P network modeled by a directed graph $G = (V, E)$ with *bounded tree degree*. That is, we bound the number of children a node can have in individual trees, although across multiple trees the overall node degree can be unbounded. As shown later, this variant problem is still challenging to solve in general, and we are looking for polynomial-time approximation solutions.

Let T_s be the set of feasible spanning/Steiner trees over G rooted at source s and reaching all receiver peers in R_s, and all these trees have tree degree bound M. Let y_t be the rate of tree $t \in T_s$. We formulate the single source streaming capacity problem as a Linear Programming (LP) problem. However, the number of variables is clearly exponential in the network size, and standard LP solutions cannot be applied to obtain polynomial time solutions.

Single Source Primal Problem (Maximum Streaming Rate)

$$\max_{y_t \geq 0, \forall t \in T_s} \quad \sum_{t \in T_s} y_t \tag{24}$$

$$\text{s.t.} \quad \sum_{t \in T_s} m_{v,t} y_t \leq C_{out}(v), \forall v \in V \tag{25}$$

---

where $m_{v,t}$ is the number of outgoing edges of node v in tree t, and (25) corresponds to the upload capacity constraints.

From linear programming duality theory, solving the primal problem is equivalent to solving its dual problem, and an optimizer of the dual problem readily leads to an optimizer of the primal algorithm. The dual problem associates a non-negative variable (interpreted as price) $p(v)$ with each node $v \in V'$ corresponding to constraint (25), and can be written as follows:

Single Source Dual Problem (Min-Cost Network)

$$\min_{p(v)\geq 0,\forall v\in V} \quad \sum_{v\in V} C_{out}(v)p(v) \tag{26}$$

$$\text{s.t.} \quad \sum_{v\in V} m_{v,t}p(v) \geq 1\,, \forall t \in T \tag{27}$$

In general, a network can have an exponential number of trees (in the size of the network) with a common tree degree bound. Hence, the primal problem can have possibly exponential number of *variables* and its dual can have an exponential number of *constraints*, and are both are hard to solve.

Inspired by the seminal work by Garg and Konemann [22] on solving multi-commodity flow problem using an iterative algorithm, our work [21] designs a fast (polynomial time) combinatorial algorithm for solving both primal and dual problems up to an approximation factor. Note Garg-Konemann approach was also adopted by Cui et al. in [23] to develop a primal-dual algorithm to maximize broadcast rate over undirected graph with edge capacity constraints, no helper node, and no node degree bound.

One observation that can be made from our approach in [21] is that although solving the **Single Source Primal/Dual** problems exactly requires packing exponential number of trees, solving them approximately only requires packing polynomial number of trees. We find this set of polynomial number of trees and assigning the corresponding tree rates in an iterative manner in [21]. We give a briefing description as below:

1) Find a min-cost tree over the node-costed network with given node cost, and assign a tree rate that is subjected to normalization in Step 3.
2) Include the newly founded min-cost tree into the candidate tree set, and update the cost of the nodes that is used by this newly found tree and its assigned tree rate. If the overall network can accommodate more flows (by testing whether the overall network cost is too large), then go to Step 1; otherwise go to Step 3. The intuition of updating the node costs is to reflect the usage of the nodes via its costs, thus the nodes will be less likely to be picked when computing the min-cost tree with the updated node costs.
3) Use only the trees in the candidate tree set collected in Step 2, and normalize their tree rates. It can be shown that at the end we have find a set of polynomial number of trees with corresponding tree rates to solve the **Single Source Primal/Dual** Problems approximately.

The key step in our iterative approach involves finding *the min-cost tree* over (directed) node-costed networks that have *costs on nodes rather than on edges*.

Theorem 5: If an α-approximation algorithm ($\alpha \geq 1$) exists for the min-cost tree problem, then our algorithm can guarantee an approximation factor of $(\alpha + \epsilon)$ for any $\epsilon > 0$.

Hence, approximation algorithms for the min-cost tree problem, when plugged into our primal-dual framework, give approximation algorithms for the overall problem.

The min-cost (spanning/Steiner) tree problem over node-costed networks is essentially the minimum cost directed Steiner tree problem *together with symmetric connectivity and a special structure on the costs – costs of all edges going out of a node are equal*. On one hand, one can directly apply approximation algorithms for general problems to this specific case to obtain approximated solution for the overall problem. Our work in [21] includes a summary of the results obtained along this path.

On the other hand, such special structure can in fact be utilized to design algorithm with improved approximation factor, as compared to those for general problems without such special structure.

For instance, it is known that the minimum cost directed Steiner tree problem over edge-costed networks is hard to approximate to a factor better than $\ln |R_s|$ [24]. An $O\left(|R_s|^{\delta}\right)$-factor approximation algorithm that runs in polynomial time for any fixed $\delta > 0$ is given in [25]. By reducing the min-cost Steiner tree problem over node-costed networks to a min-cost group Steiner tree problem over edge-costed network, we shows in [21] that an approximation factor of $O(\ln |R_s| \ln^3 |V|)$ can be achieved in polynomial time. In the case where $|V| = O(|R_s|)$, the approximation factor becomes $O\left(\ln^4 |R_s|\right)$, which is better than the best known approximation factor of $O(|R_s|^{\delta})$ ($\delta > 0$) over traditional edge-costed networks [26].

E. Simulations

1) Evaluation on Large-Scale Online Video Viewing Trace: In this simulation, we evaluate the ability of Cluster-Tree algorithm, the only distributed algorithm surveyed here and the one capable of handling per-node degree bound, to provider a sustained streaming rate as peers join over time. For this purpose, we use an MSN VoD (Video-on-Demand) trace from [18] – 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. 8, 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.

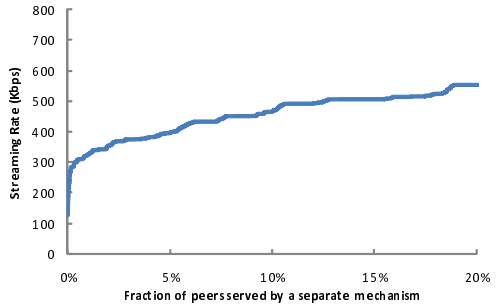

Fig. 8. Streaming rate (Kbps) for Cluster-Tree scheme as a function of the fraction of peers (the lowest uplink capacity ones) served by a separate delivery mechanism. (MSN online video trace used for simulation.)

We therefore propose to complement the theory-based algorithm with 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. Various real-time measurement techniques can readily help identify such links with little overhead. To validate that this works, we demonstrate, in Fig. 8, the sustained streaming rate in the Cluster-Tree scheme as peers join over time, when a small fraction (we call 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. 9, 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. 9, a higher cutoff percentage gives a higher streaming rate. For example, we can get a higher streaming rate of about 400 Kbps by increasing the cutoff to 5% of the peers. The theoretical maximum streaming rate (computed as the global average uplink capacity μ) is also shown.

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 complementary delivery mechanism is used for the ultra-low uplink capacity peers.

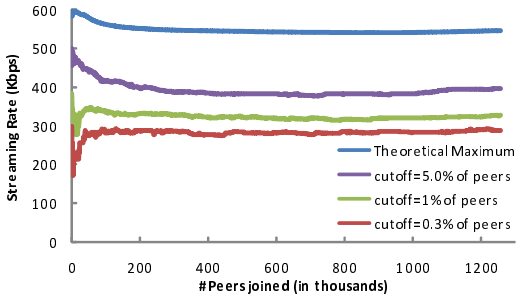

Fig. 9. Sustained streaming rate (Kbps) for Cluster-Tree scheme as peers arrive over time. Different fixed fractions of peers (the lowest uplink capacity ones) are served by a separate delivery mechanism on a continuing basis. Theoretical maximum streaming rate is also shown. (MSN online video trace used for simulation.)

III. Open Problems

A. An Open Case

When the given graph is not full mesh, each node has a degree bound (even just per-tree degree bound), and there are helper nodes, polynomial time computation of P2P streaming capacity with a constant approximation fraction remains open.

B. The Cases with Both Upload and Download Capacity Constraints

We observe that a graph G with both node upload and download capacity constraints can be reduced to a corresponding graph G' with only upload capacity constraints. Given a graph $G = (V, E)$ with nodes $v \in V$ having upload capacities $C_{out}(v)$ and download capacities $C_{in}(v)$, we can construct a graph G' by replacing each node v by two nodes v^-, v^+ as shown in Fig. 10. Let $G' = (V^- \cup V^+, E')$ where $V^- = \{v^- : v \in V\}$ and similarly for V^+, and $E' = \{(v^-, v^+) : v \in V\} \cup \{(u^+, v^-) : u, v \in V\}$. This graph G' has only upload capacity constraints given by $C_{out}(v^-) = C_{in}(v)$ and $C_{out}(v^+) = C_{out}(v)$.

Proposition 2: Suppose network coding is not allowed, then a scheme achieves maximum streaming rate on G (with download and upload constraints) if and only if it achieves maximum streaming rate on G' (with only upload constraints).

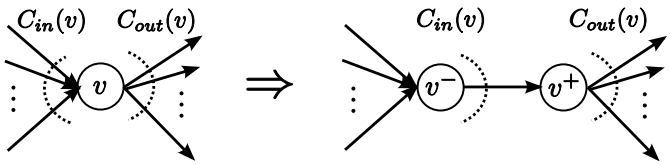

Fig. 10. A node v with upload and download capacity constraints $C_{out}(v)$ and $C_{in}(v)$ respectively is replaced with two nodes v^- and v^+ with only upload capacity constraints. Node v^- has an upload capacity of $C_{in}(v)$, and v^+ has an upload capacity of $C_{out}(v)$.

Proposition 2 indicates that if network coding is not allowed, then it is sufficient to study a graph G' with only upload capacity constraint. The main difficulty is in efficiently finding schemes to maximize streaming rate on this *non-complete* graph G'. When the original graph G has bounded node degree, the problem becomes NP-complete, as is discussed in Section II-C.

With network coding allowed, streaming rate achievable on G' is not necessarily achievable on G. The amount of traffic node v can mix on G is no more than $C_{in}(v)$, whereas the net traffic v^- and v^+ can mix on G' can exceed $C_{in}(v)$.

C. Streaming Capacity with Network Coding Allowed

Almost all existing work we review in the previous section focus on routing based algorithms. It is of interest to study the streaming capacity problem with network coding allowed. We recently characterize the capacity region of P2P networks with network coding allowed, for the cases with full-mesh networks and presence of helper [27], [28]. It is shown that network coding has no gain over routing in terms of throughput in P2P networks. This is contrast to the general belief that network coding can improve the capacity region as compared to routing. Similar results were also observed by authors in [29] for the case with full-mesh networks and no helper.

D. Distributed Algorithms and Implementations

Almost all existing work we review in the previous section use centralized algorithms, with few exceptions such as Mutualcast [4]. It is of great interest to solve the streaming capacity problem using distributed algorithms. Systems running distributed algorithms, compared with those running centralized algorithms, are more adaptable to users joining and leaving the systems (e.g., peer churn in Peer-to-peer systems) and are more robust to system/network dynamics (e.g., upload capacity fluctuations).

So far in the research literature of P2P streaming, there is a gap between theory and practice. Bridging this gap through a large-scale deployment of theory-inspired algorithms will be a highly challenging and healthy direction.

E. Additional Metrics and Variables

In addition to throughput, delay, robustness (to traffic and peer fluctuations) and ISP-friendliness are also important, and sometimes conflicting, objectives in P2P streaming. In addition to overlay topology construction and application layer streaming rate, there are other variables that present opportunities of design freedom to either P2P or ISP.

For instance, a recent work by Wu, Liu, and Ross [30] studies multi-channel P2P streaming systems. They suggest to decouple the content a peer downloads to serve and the content it wants to view, and shows throughput improvement in the presence of dynamic peer joining and leaving.

Bridging the content-pipe divide [31] and jointly optimizing over these variables to strike 4-dimensional tradeoff regions of the above metrics remains a long-term goal for this research topic.

ACKNOWLEDGEMENT

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

Part of this work has been supported by the Princeton EDGE Lab sponsored in part by NSF Computing Research Infrastructure, ONR Defense University Research Instrumentation Program, and Qualcomm. Minghua Chen is supported by Competitive Earmarked Research Grants (Project Number 411008 and 411209) established under the University Grant Committee of the Hong Kong Special Administrative Region, China, a Direct Grant (Project Number 2050397) of The Chinese University of Hong Kong, Cisco, and Microsoft. Part of the work was done when Shao Liu visited The Chinese University of Hong Kong.

REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," no. 4, pp. 1204–1216, Jul. 2000.

[2] S. Li, R. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[3] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52(10), pp. 4413–4430, Oct. 2004.

[4] J. Li, P. A. Chou, and C. Zhang, "Mutualcast: an efficient mechanism for content distribution in a p2p network," in *Proceedings of Acm Sigcomm Asia Workshop*, Beijing, China, Apr. 2005.

[5] R. Kumar, Y. Liu, and K. W. Ross, "Stochastic fluid theory for p2p streaming systems," in *Proc. IEEE INFOCOM*, Anchorage, AL, May 2007.

[6] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized Decentralized Broadcasting Algorithms," in *INFOCOM 2007*, 2007, pp. 1073–1081.

[7] T. Nguyen, K. Kolazhi, R. Kamath, S. Cheung, and D. Tran, "Efficient Multimedia Distribution in Source Constraint Networks," *Multimedia, IEEE Transactions on*, vol. 10, no. 3, pp. 523–537, 2008.

[8] J. Edmonds, "Edge-disjoint branchings," *Combinatorial Algorithms, R. Rustin, ed.*, pp. 91–96, 1973.

[9] Pplive - internet peer-to-peer video streaming. [Online]. Available: http://www.pplive.com

[10] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale P2P IPTV system," *IEEE Transactions on Multimedia*, vol. 9, no. 8, 2007.

[11] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance bounds for peer-assisted live streaming," in *Proceedings of the 2008 ACM SIGMETRICS*, 2008, pp. 313–324.

[12] S. Liu, M. Chiang, S. Sengupta, J. Li, and P. A. Chou, "Snowball algorithm for P2P capacity computation," in *46th Allerton Conference on Communication, Control and Computing*, 2008.

[13] V. N. Padmanabhan and K. Sripanidkulchai, "The case for cooperative networking," in *Proceedings of IPTPS '02*. Springer, 2002.

[14] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in cooperative environments," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM New York, NY, USA, 2003, pp. 298–313.

[15] S. Liu, M. Chen, M. Chiang, S. Sengupta, J. Li, and P. A. Chou, "P2p streaming capacity under node degree bound," *Technical Report*, 2009, available at http://www.princeton.edu/∼chiangm/infocom10long.pdf.

[16] PPStream. http://www.ppstream.com/.

[17] 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.

[18] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" in *Proc. ACM SIGCOMM*, Kyoto, Japan, August 2007.

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

[20] J. Lenstra, A. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Studies in Integer Programming*, vol. 1, pp. 343–362, 1977.
[21] S. Sengupta, S. Liu, M. Chen, M. Chiang, J. Li, and P. A. Chou, "Streaming Capacity in Peer-to-Peer Networks with Tree Degree Constraints," 2009, submitted to IEEE Trans. on Information Theory.
[22] N. Garg and J. Konemann, "Faster and simpler algorithms for multicommodity flow and otherfractional packing problems," in *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, 1998, pp. 300–309.
[23] Y. Cui, B. Li, and K. Nahrstedt, "On achieving optimized capacity utilization in application overlay networks with multiple competing sessions," in *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*. ACM New York, NY, USA, 2004, pp. 160–169.
[24] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM*, vol. 45, no. 4, pp. 634–652, 1998.
[25] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li, "Approximation algorithms for directed Steiner problems," in *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, 1998, pp. 192–200.
[26] M. Charikar and A. Agarwal, "On the Advantage of Network Coding for Improving Network Throughput," in *Proceedings of the IEEE Information Theory Workshop*, 2004.
[27] S. Sengupta, M. Chen, P. Chou, and J. Li, "On Optimality of Routing for Multi-source Multicast Communication Scenarios with Node Uplink Constraints," in *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, 2008, pp. 330–334.
[28] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility maximization in p2p systems," in *Proc. ACM SIGMETRICS*, Annapolis, Maryland, USA, Jun. 2008.
[29] D. Chiu, R. Yeung, J. Huang, and B. Fan, "Can network coding help in p2p networks," in *NetCod*.
[30] D. Wu, Y. Liu, and K. W. Ross, "Queueing Network Models for Multi-Channel P2P Live Streaming Systems," in *Proceedings of IEEE Infocom*, 2009.
[31] M. Chiang, "Content-pipe divide and network distribution capacity," in *Proc. CISS*, Princeton, NJ, March 2008.