

On Achieving Optimized Capacity Utilization in Application Overlay Networks with Multiple Competing Sessions

Yi Cui

Dept. of Computer Science
University of Illinois at Urbana-Champaign
{yicui}@cs.uiuc.edu

Baochun Li

Dept. of Electrical and Computer Engineering
University of Toronto
{bli}@eecg.toronto.edu

Klara Nahrstedt

Dept. of Computer Science
University of Illinois at Urbana-Champaign
{klara}@cs.uiuc.edu

ABSTRACT

In this paper, we examine the problem of large-volume data dissemination via overlay networks. A natural way to maximize the throughput of an overlay multicast session is to split the traffic and feed them into multiple trees. While in single-tree solutions, bandwidth of leaf nodes may remain largely under-utilized, multi-tree solutions increase the chances for a node to contribute its bandwidth by being a relaying node in at least one of the trees. We study the following problems: (1) What is the maximum capacity multi-tree solutions can exploit from overlay networks? (2) When multiple sessions compete within the same network, what is the relationship of two contradictory goals: achieving fairness and maximizing overall throughput? (3) What is the impact of IP routing in achieving at constraining the optimal performance of overlay multicast?

We extend the multicommodity flow model to the case of overlay data dissemination, where each commodity is associated with an overlay session, rather than the traditional source-destination pair. We first prove that the problem is solvable in polynomial time, then propose an ϵ -approximation algorithm, assuming that each commodity can be split in arbitrary ways. The solution to this problem establishes the theoretical upper bound of overall throughput that any multi-tree solution could reach. We then study the same problem with the restriction that each commodity can only be split and fed into a limited number of trees. A randomized rounding algorithm and an online tree-construction algorithm are presented. All these algorithms are evaluated by extensive simulations.

Categories and Subject Descriptors

C.2.5 [Local and Wide-Area Networks]: Internet; D.4.8 [Performance]: Simulation; G.1.6 [Optimization]: Linear Programming, Integer Programming; G.2.2 [Graph Theory]: Trees

General Terms

Algorithms, Measurement, Performance, Theory

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SPAA'04, June 27-30, 2004, Barcelona, Spain.

Copyright 2004 ACM 1-58113-840-7/04/0006...\$5.00.

Keywords

Overlay, Multicommodity Flow, Multicast

1. INTRODUCTION

High-bandwidth data dissemination at the application layer has recently emerged as an important research topic [1, 2, 3], in order to realize the tremendous potential of application-layer overlay networks. In general, the prevailing trend in recent research is to realize data dissemination from one data source to multiple receivers via the construction of a *multicast tree* at the application layer, as a replacement of traditional IP multicast, which is not widely available in the IP backbone. The general approach common to all existing proposals is to organize end systems into a logical overlay network, and to transfer data along the edges of such an overlay network using unicast transport services. As one of the key benefits, the application layer offers unprecedented flexibility and freedom to design algorithms that incorporate a variety of Quality-of-Service considerations, and bandwidth is one of the most important metrics to be considered.

In order to achieve better utilization of underutilized network capacities when disseminating data via overlay multicast, it is natural to resort to the construction of multiple concurrent multicast trees. In this approach, data is split to multiple slices (e.g., using source erasure codes), and each slice is transmitted along one of the multicast trees [2, 3, 4]. The intuition is that, while in single-tree solutions, bandwidth of leaf nodes may remain largely under-utilized, the multi-tree approach increases the chances for a node to contribute its bandwidth by being a non-leaf node in at least one of the trees [2]. Though this intuition shows promise, it is nevertheless unclear with respect to the number of trees that needs to be constructed in order to maximize *capacity utilization* and *end-to-end throughput*, and what is the optimal way to construct these trees. These open problems are exacerbated when multiple dissemination sessions may co-exist in the network, each consisting of a different data source, as well as multiple trees to a different group of receivers. In the case of multiple sessions, the issue of inter-session *fairness* needs to be considered when maximizing capacity utilization.

In this paper, we seek to analytically and experimentally investigate the complete spectrum of such a multi-tree design philosophy, especially when multiple data dissemination sessions co-exist. Our objective is simple: we prefer to design algorithms that may maximize the *end-to-end throughput* for all co-existing sessions in an overlay network, and maximally exploit the capacity an overlay network has to offer. It may be shown that this problem is far from trivial. As examples, the following questions may naturally arise. First, for a given multicast session, what is the maximum capacity it can exploit from the overlay network, and how many multi-

cast trees are needed to achieve such maximum? Second, when we seek to optimize the utilization of overlay network capacities, will there be an inherent incompatibility between capacity utilization and inter-session fairness? Third, can we design an efficient and online algorithm to approximate the theoretical upper bound with a very limited number of trees in each session? Finally, what is the impact of IP routing when we seek answers to all the previous questions? Do different IP routing strategies adversely affect the achievable end-to-end throughput of overlay multicast?

We provide analytical and experimental insights towards addressing these important questions, and propose an extensive array of approximation algorithms to achieve the best possible capacity utilization, with multiple trees in each dissemination session. Our proposed algorithms are progressively more realistic as they are unveiled, and the effectiveness of our proposed algorithms is verified using extensive simulations, some of which are interleaved with our theoretical discussions.

The remainder of this paper is organized as follows. Sec. 2 presents our theoretical foundation based on multicommodity flows. Sec. 3 presents an array of combinatorial approximation algorithms to the problems of maximizing capacity utilization in overlay networks. Sec. 4 brings further reality into consideration, and proposes online algorithms to address the *unsplittable flow* problem, where data flows can only be split into a specified number of sub-flows of fixed rates. Sec. 5 quantitatively evaluates the impact of IP routing strategies with respect to limiting the optimal capacity utilization of overlay multicast. Sec. 6 and 7 discuss related work and conclude the paper.

2. MODEL

The problem of achieving maximum capacity utilization among competing overlay sessions can be understood as a multicommodity flow problem. The data to be disseminated within an overlay session can be considered as its commodity. Each session expects to maximize the throughput of its own commodity.

2.1 Multicommodity Flow Problem: a Review

We first review the typical multicommodity flow problem in the setting of source-destination pairs. Let $G = (V, E)$ be an undirected graph, with capacity c_e on each edge $e \in E$. We are given k commodities, K_1, K_2, \dots, K_k . Each commodity is a tuple $K_i = ((s_i, d_i), dem(i))$. Here, s_i is the source, and d_i is the destination of K_i . $dem(i)$ is the demand of K_i , which is the desired flow value for K_i from s_i to d_i . In addition, a set of paths exist between s_i and d_i , denoted as $\mathcal{P}_i = \{p_j^i\}$. Each commodity can be arbitrarily split and sent along several paths in parallel. We use f_j^i to denote the flow of commodity K_i sent along the path p_j^i . We further introduce a 0-1 variable $n_e(p_j^i)$. $n_e(p_j^i) = 1$ if e appears in the path p_j^i . Otherwise, $n_e(p_j^i) = 0$. The objective is to maximize the overall flows of all commodities, subject to the flow conservation and capacity constraints. Using the linear programming (LP) formulation, we have

$$\begin{aligned} \mathbf{P1} : \quad & \text{maximize} && \sum_{i=1}^k \sum_{j=1}^{|\mathcal{P}_i|} f_j^i && (1) \\ & \text{subject to} && \sum_{i=1}^k \sum_{j=1}^{|\mathcal{P}_i|} n_e(p_j^i) \cdot f_j^i \leq c_e, \forall e \in E \\ & && f_j^i \geq 0, \forall i, \forall j \end{aligned}$$

We refer to **P1** as the *maximum flow problem*. However, **P1**

does not consider the issue of fairness. In the following alternative problem formulation, the objective is to maximize f , referred to as *throughput*, such that for each commodity K_i , at least $f \cdot dem(i)$ units of commodity flow can be routed simultaneously, subject to the flow conservation and capacity constraints.

$$\begin{aligned} \mathbf{P2} : \quad & \text{maximize} && f && (2) \\ & \text{subject to} && \sum_{j=1}^{|\mathcal{P}_i|} f_j^i \geq f \cdot dem(i), i = 1, \dots, k \\ & && \sum_{i=1}^k \sum_{j=1}^{|\mathcal{P}_i|} n_e(p_j^i) \cdot f_j^i \leq c_e, \forall e \in E \\ & && f \geq 0, f_j^i \geq 0, \forall i, \forall j \end{aligned}$$

We refer to **P2** as the *maximum concurrent flow problem*. **P2** enforces fairness by requiring that the comparative ratio of traffic routed for different commodities satisfies the comparative ratio of their demands. Thus, the absolute value of $dem(i)$ is meaningless, as we can easily tune the value of f by scaling up/down all demands, while $f \cdot dem(i)$ stays unchanged.

2.2 Problem Formulation

Now consider the overlay multicast version of **P1** and **P2**. Here, each commodity is redefined as $K_i = (S_i, dem(i))$, $i = 1, \dots, k$, where S_i is an overlay multicast session consisting of a set of vertices. We define $\mathcal{T}_i = \{t_j^i\}$ as the set of all overlay trees, each of which covers all vertices in S_i . We here reuse f_j^i to denote the flow of commodity K_i sent along the tree t_j^i . We also use $n_e(t_j^i)$ to represent the appearance of e in t_j^i . In this case, $n_e(t_j^i)$ could be an integer greater than one, since a physical edge e may appear in t_j^i more than once. Therefore, $n_e(t_j^i)$ denotes the number of times e appears in t_j^i . In this context, the counterpart of **P1** is

$$\begin{aligned} \mathbf{M1} : \quad & \text{maximize} && \sum_{i=1}^k \sum_{j=1}^{|\mathcal{T}_i|} \frac{|S_i| - 1}{|S_{max}| - 1} \cdot f_j^i && (3) \\ & \text{subject to} && \sum_{i=1}^k \sum_{j=1}^{|\mathcal{T}_i|} n_e(t_j^i) \cdot f_j^i \leq c_e, \forall e \in E \\ & && f_j^i \geq 0, \forall i, \forall j \end{aligned}$$

Since for each session S_i , there are $|S_i| - 1$ receivers, $(|S_i| - 1) \cdot f_j^i$ is the aggregate flow of the entire session. S_{max} is the session with the most number of receivers. Thus, **P1** can be understood as a special case of **M1**, where each session consists of only one receiver, i.e., $|S_i| - 1 = 1$.

The counterpart of **P2** is

$$\begin{aligned} \mathbf{M2} : \quad & \text{maximize} && f && (4) \\ & \text{subject to} && \sum_{j=1}^{|\mathcal{T}_i|} f_j^i \geq f \cdot dem(i), i = 1, \dots, k \\ & && \sum_{i=1}^k \sum_{j=1}^{|\mathcal{T}_i|} n_e(t_j^i) \cdot f_j^i \leq c_e, \forall e \in E \\ & && f \geq 0, f_j^i \geq 0, \forall i, \forall j \end{aligned}$$

The problems **M1** and **M2** can not be directly addressed, since there exists an exponential number of constraints. For each session

S_i , the number of possible overlay trees is exponential, i.e., $|\mathcal{T}_i| = |S_i|^{|S_i|-2}$, by Cayley's theorem [5]. However, **M1** and **M2** are still solvable if we can find a separation oracle [6] — a polynomial algorithm — to verify whether a given solution to **M1** or **M2** is feasible. Before introducing such an algorithm, we first discuss the following problem.

2.3 Packing Spanning Trees

Suppose a session S_i has vertices $\{v_1^i, \dots, v_{|S_i|}^i\}$. Let $p(v_m^i, v_n^i)$ be the unicast route¹ between v_m^i and v_n^i , $f(v_m^i, v_n^i)$ the total amount of traffic between v_m^i and v_n^i within session S_i . We construct a complete graph $G_i = (S_i, E_i)$, in which the weight of an edge $(v_m^i, v_n^i) \in E_i$ is $f(v_m^i, v_n^i)$. We are interested with the problem of how to decompose G_i into a set of spanning trees, such that their aggregate rates maximally saturate the capacity of G_i . Consider an

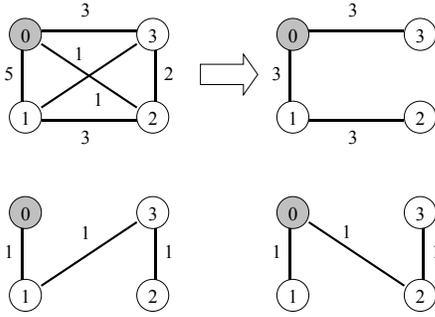


Figure 1: Packing Spanning Trees

example in Fig. 1. The overlay session has 4 nodes. Node 0 is the source. The weight of each edge is the total amount of traffic between its two nodes. In this example, the session can be decomposed into 3 overlay trees, whose aggregate rate is 5. This problem is formalized as

$$\begin{aligned} \mathbf{S} : \quad & \text{maximize} \quad \sum_{j=1}^{|\mathcal{T}_i|} f_j^i \\ & \text{subject to} \quad \sum_{(v_m^i, v_n^i) \in E_i} f_j^i \leq f(v_m^i, v_n^i), \forall (v_m^i, v_n^i) \in E_i \\ & \quad \quad \quad f_j^i \geq 0, \forall i, \forall j \end{aligned} \quad (5)$$

This problem is known as “packing spanning trees”. A min-max relation for **S** has been given by Tutte [7] and Nash-Williams [8], as follows. Let $\delta(G_i)$ be a partition of G_i . $|\delta(G_i)|$ is the number of separate components it results to. $f(\delta(G_i))$ is the weight sum of all edges across $\delta(G_i)$, formally defined as

$$f(\delta(G_i)) = \sum_{v_m^i \text{ and } v_n^i \text{ belong to different components of } \delta(G_i)} f(v_m^i, v_n^i).$$

The maximum of **S** is the minimum of

$$\frac{f(\delta(G_i))}{|\delta(G_i)| - 1} \quad (6)$$

over all partitions of G_i . Cunningham [9] gave a polynomial algorithm for finding the minimum of (6), and the set of trees to achieve this minimum, by reducing it to $|S_i||E_i|$ maximum flow problems. Barahona [10] showed that the same problem can be reduced to

¹This route is determined by IP-level routing.

$|S_i|^2$ maximum flow problems. Both algorithms can be employed as the separation oracle to the following reformulation of **M1** and **M2**.

2.4 Problem Reformulation

$$\begin{aligned} \mathbf{M1}' : \quad & \text{maximize} \quad \sum_{i=1}^k \frac{|S_i| - 1}{|S_{max}| - 1} \cdot \min \left(\frac{f(\delta(G_i))}{|\delta(G_i)| - 1} \right) \\ & \text{subject to} \quad \sum_{i=1}^k \sum_{e \in p(v_m^i, v_n^i)} f(v_m^i, v_n^i) \leq c_e, \forall e \in E \\ & \quad \quad \quad f(v_m^i, v_n^i) \geq 0, \forall i, \forall (v_m^i, v_n^i) \in G_i \end{aligned} \quad (7)$$

$$\begin{aligned} \mathbf{M2}' : \quad & \text{maximize} \quad f \\ & \text{subject to} \quad \min \left(\frac{f(\delta(G_i))}{|\delta(G_i)| - 1} \right) \geq f \cdot \text{dem}(i), i = 1, \dots, k \\ & \quad \quad \quad \sum_{i=1}^k \sum_{e \in p(v_m^i, v_n^i)} f(v_m^i, v_n^i) \leq c_e, \forall e \in E \\ & \quad \quad \quad f \geq 0, f(v_m^i, v_n^i) \geq 0 \quad \forall i, \forall (v_m^i, v_n^i) \in G_i \end{aligned} \quad (8)$$

With **M1'** and **M2'**, we reduce the number of constraints for each session S_i from exponential to $O(|S_i|^2)$. **M1'** and **M2'** can be solved by standard LP solving techniques such as ellipsoid method [6].

3. DERIVING OPTIMAL CAPACITY UTILIZATION IN OVERLAY NETWORKS

Although **M1** and **M2** are proved solvable, finding their exact solutions by ellipsoid method can be slow and expensive. Instead, we are interested to find a fully polynomial time approximation scheme (FPTAS) to these problems. A FPTAS is a family of algorithms that finds an ϵ -approximate solution, which returns a result at least $(1 - \epsilon)$ times the maximum value, for any error parameter $\epsilon > 0$. Its running time is polynomial in the size of the network ($|V|$ and $|E|$), the number of commodities (k), and $1/\epsilon$. In this section, we propose a FPTAS to **M1** and **M2** based on the scheme proposed by Garg and Konemann [11], which was later improved by Fleischer [12]. The proofs of all lemmas and theorems in this section can be found in our technical report [13].

3.1 Algorithm for the Maximum Flow Problem

Before presenting our algorithm for **M1**, we first formulate its dual as follows.

$$\begin{aligned} \mathbf{D1} : \quad & \text{minimize} \quad \sum_{e \in E} c_e \cdot d_e \\ & \text{subject to} \quad \sum_{e \in E} n_e(t_j^i) \cdot d_e \geq \frac{|S_i| - 1}{|S_{max}| - 1}, t_j^i \in \mathcal{T}_i, i = 1, \dots, k \\ & \quad \quad \quad d_e \geq 0, \forall e \in E \end{aligned} \quad (9)$$

D1 corresponds to the problem of assigning length d_e to each edge $e \in E$, such that the length of any spanning tree in \mathcal{T}_i ($i = 1, \dots, k$) is at least $\frac{|S_i|-1}{|S_{max}-1}$. By LP duality theory [6], the minimum of **D1** is the maximum of **M1**. Here, d_e represents the marginal cost of using an additional unit of capacity of e .

MaxFlow	
1	$\forall e \in E, d_e \leftarrow \beta$
2	$f_j^i \leftarrow 0, t_j^i \in \mathcal{T}_i, i = 1, \dots, k$
3	loop
4	for $i = 1$ to k do
5	$t^i \leftarrow$ minimum overlay spanning tree in \mathcal{T}_i using d_e
6	$minlen \leftarrow \min_{i=1}^k \sum_{e \in E} n_e(t^i) \cdot d_e \frac{ S_{max} -1}{ S_i -1}$
7	$t \leftarrow \arg \min_{i=1}^k \sum_{e \in E} n_e(t^i) \cdot d_e \frac{ S_{max} -1}{ S_i -1}$
8	if $minlen \geq 1$
9	return
10	$c \leftarrow \min_{e \in t} \frac{c_e}{n_e(t)}$
11	$f(t) \leftarrow f(t) + c$
12	$\forall e \in t, d_e \leftarrow d_e(1 + \epsilon \frac{n_e(t)c}{c_e})$
13	end loop

Table 1: Algorithm for the Maximum Flow Problem

The algorithm for the *maximum flow* problem, henceforth referred to as **MaxFlow**, is shown in Table 1. Initially, we set $d_e = \beta$ for each edge $e \in E$, and $f_j^i = 0$ for each tree t_j^i in each session S_i . In each iteration, a “minimum overlay spanning tree” t^i is computed for each session S_i as follows. We first construct an overlay graph for S_i , a complete graph $G_i = (S_i, E_i)$. Each edge $(v_m^i, v_n^i) \in E_i$ corresponds to the unicast route between v_m^i and v_n^i , $p(v_m^i, v_n^i)$. Straightforwardly, the length of (v_m^i, v_n^i) is the sum of lengths of all edges along $p(v_m^i, v_n^i)$. Then we can obtain t^i by running the minimum spanning tree algorithm on G_i . We proceed to choose t among all t^i , whose normalized length $\sum_{e \in E} n_e(t^i) \cdot d_e \frac{|S_{max}|-1}{|S_i|-1}$ is the minimum. We check if its cost is no less than 1. If so, it means that the lengths of all spanning trees are no less than 1, then we stop the algorithm. Otherwise, we send c units of traffic along t , which is the bottleneck capacity of t . Since at most $\frac{c_e}{n_e(t)}$ units of traffic of t can be sent through e , c is $\min_{e \in t} \frac{c_e}{n_e(t)}$. Finally, for each edge e going through t , d_e is augmented by the factor $1 + \epsilon \frac{n_e(t)c}{c_e}$. Following the same way as Garg and Konemann [11], we prove the following sequence of lemmas.

Lemma 1: **MaxFlow** terminates after at most $|E| \log_{1+\epsilon} \frac{1+\epsilon}{\beta}$ iterations, S_{max} being the session of the maximum size.

Lemma 2: Scaling the final flow by $\log_{1+\epsilon} \frac{1+\epsilon}{\beta}$ yields a feasible primal solution.

Lemma 3: When $\beta = \frac{(1+\epsilon)^{1-1/\epsilon}}{[U(|S_{max}|-1)U]^{1/\epsilon}}$, the final flow scaled by $\log_{1+\epsilon} \frac{1+\epsilon}{\beta}$ has a value at least $(1-2\epsilon)$ times the optimal value of M1. U is the length of the longest unicast route.

Each iteration of the algorithm involves k minimum overlay spanning tree operations. Regarding the running time, we have the following theorem.

Theorem 1: When $\beta = \frac{(1+\epsilon)^{1-1/\epsilon}}{[U(|S_{max}|-1)U]^{1/\epsilon}}$, the running time is $O(\frac{k|E|}{\epsilon^2} [\log U + \log(|S_{max}|-1)]) \cdot T_{mst}$. U is the length of the longest unicast route and T_{mst} is the running time of the minimum overlay spanning tree construction algorithm.

Now we calculate T_{mst} . The running time of Prim’s algorithm is $m + n \log n$, n being the number of vertices and m the number of edges. Since the overlay graph G_i is a complete graph, $T_{mst} = \frac{|S_i|(|S_i|-1)}{2} + |S_i| \log |S_i|$, which is $O(|S_{max}|^2)$, S_{max} being the

session of the maximum size.

3.2 An Experiment (Part One)

We conduct a simple experiment to illustrate how our algorithm works. Using the Boston BRITE topology generator, we create a 100-node router-level topology by the Waxman model. All edges have capacities of 100. Two multicast sessions are randomly created over this topology. Session 1 has 7 nodes, session 2 has 5 nodes. They have the same demand as 100. The unicast path between any pair of nodes with each session is determined by shortest-path routing.

Table 2 shows the result of **MaxFlow** with different approximation ratios. The *overall throughput* is the aggregate receiving rate of all session members, i.e., (Rate of Session 1) $\cdot 6 +$ (Rate of Session 2) $\cdot 4$. From the data, we have the following observations. First, the calculated optimal throughput slightly increases as we tighten the approximation ratio. Second, the number of trees needed to achieve it also increases with a trend of accelerated speed, although not always increasing. We notice that session 1 has more trees than session 2 does. However, considering the exponential growth of the solution space ($|S_i|^{|S_i|-2}$ possible trees for session S_i), this number is only a small portion (397 out of $7^5 = 16807$, 2.36%), compared to the same value for session 2 (44 out of $5^3 = 125$, 35.2%). Third, the running time of the algorithm grows quadratically as the approximation ratio increases (recall the $\frac{1}{\epsilon^2}$ factor in the running time analysis, shown in Theorem 1). Finally, the rate of session 1 is much greater than session 2. This is because the nature of **MaxFlow** (maximizing overall throughput) makes it to prefer the session of a larger size, since increasing the rate of session 1 by a certain amount brings more benefits than doing the same to session 2. This naturally leads to the algorithm for maximum concurrent flow problem in the next subsection, which considers the issue of fairness.

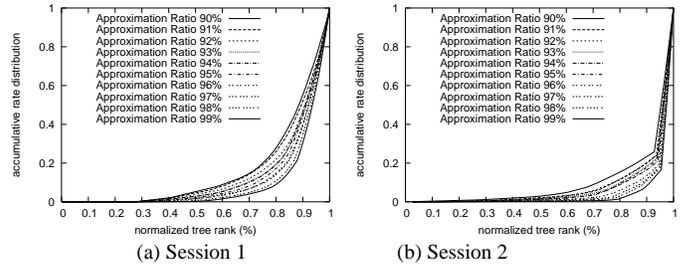


Figure 2: Overlay Tree Rate Distribution (MaxFlow)

Another interesting observation may be drawn from Fig. 2, which plots the accumulative rate distribution among all overlay trees for session 1 and 2. In both figures, 90% of the throughput is concentrated in less than 10% of the trees. We refer to this phenomenon as *asymmetric rate distribution*. We will discuss more on this issue towards the end of this section.

3.3 Algorithm for the Maximum Concurrent Flow Problem

Approximation Ratio	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
Rate of Session 1	163.00	163.53	163.81	164.34	164.60	164.95	165.27	165.62	165.97	166.32
Rate of Session 2	93.16	93.04	93.14	93.18	93.20	93.25	93.26	93.28	93.31	93.32
Overall Throughput	1350.63	1353.38	1355.44	1358.75	1360.45	1362.69	1364.64	1366.83	1369.07	1371.22
Number of Trees in Session 1	210	220	252	273	271	291	319	323	363	397
Number of Trees in Session 2	28	33	31	34	35	32	37	40	42	44
Running Time (number of MST operations)	2940	3606	4552	5916	8008	11482	17856	31620	70926	282266

Table 2: Experiment Result of MaxFlow

Again, we first formulate the dual of **M2** as follows.

$$\begin{aligned}
\mathbf{D2} : \quad & \text{minimize} \quad \sum_{e \in E} c_e \cdot d_e & (10) \\
& \text{subject to} \quad \sum_{e \in E} n_e(t_j^i) \cdot d_e \geq l_i, t_j^i \in \mathcal{T}_i, i = 1, \dots, k \\
& \quad \sum_{i=1}^k l_i \cdot \text{dem}(i) \geq 1 \\
& \quad d_e \geq 0, \forall e \in E, l_i \geq 0, i = 1, \dots, k
\end{aligned}$$

D2 corresponds to the problem of assigning length d_e to each edge $e \in E$ and weight l_i to each session S_i , such that for S_i , the length of any spanning tree in \mathcal{T}_i is at least l_i , and the weighted sum of l_i by $\text{dem}(i)$ over all sessions is at least 1. By LP duality theory [6], the minimum of **D2** is the maximum of **M2**. Here, d_e represents the marginal cost of using an additional unit of capacity of e , and l_i represents the marginal cost of not satisfying another unit of demand of S_i .

MaxConcurrentFlow	
1	$\forall e \in E, d_e \leftarrow \beta/c_e$
2	$f_j^i \leftarrow 0, t_j^i \in \mathcal{T}_i, i = 1, \dots, k$
3	while $\sum_{e \in E} c_e \cdot d_e < 1$
4	for $i = 1$ to k do
5	$\text{dem}'(i) \leftarrow \text{dem}(i)$
6	while $\sum_{e \in E} c_e \cdot d_e < 1$ and $\text{dem}'(i) > 0$
7	$t \leftarrow$ minimum overlay spanning tree in \mathcal{T}_i using d_e
8	$c \leftarrow \min\{\text{dem}'(i), \min_{e \in t} \frac{c_e}{n_e(t)}\}$
9	$\text{dem}'(i) \leftarrow \text{dem}'(i) - c$
10	$f(t) \leftarrow f(t) + c$
11	$\forall e \in t, d_e \leftarrow d_e(1 + \epsilon \frac{n_e(t)c}{c_e})$
12	end while
13	end for

Table 3: Algorithm for Maximum Concurrent Flow Problem

The algorithm for the *maximum concurrent flow* problem, henceforth referred to as **MaxConcurrentFlow**, is shown in Table 3. Initially, we set $d_e = \beta/c_e$ for each edge $e \in E$, and $f_j^i = 0$ for each tree t_j^i in each session S_i . The algorithm proceeds in phases. In each phase, there are k iterations. In iteration i , the objective is to route $\text{dem}(i)$ units of flow inside S_i . This is done in steps. In one step, an “minimum overlay spanning tree” t is computed the same way as in **MaxFlow**. We then send along t the amount of traffic equal to its bottleneck capacity. If the bottleneck capacity already exceeds the remaining demand $\text{dem}'(i)$, we only send $\text{dem}'(i)$ along t . Finally, for each edge e going through t , d_e is augmented the same way as in **MaxFlow**. The entire procedure stops when the

objective function value of **D2** is at least one: $\sum_{e \in E} c_e \cdot d_e \geq 1$. Following the same way as Garg and Konemann [11], we prove the following sequence of lemmas. Here, f^* is the result returned by the algorithm. OPT is the optimal value of **D2** as well as **M2**.

Lemma 4: If $OPT \geq 1$, scaling the final flow by $\log_{1+\epsilon} 1/\beta$ yields a feasible primal solution of value $f^* = \frac{t-1}{\log_{1+\epsilon} 1/\beta}$, t being the number of phases the algorithm takes to stop.

Lemma 5: If $OPT \geq 1$, then the final flow scaled by $\log_{1+\epsilon} 1/\beta$ has a value at least $(1 - 3\epsilon)$ times OPT , when $\beta = (|E|/(1 - \epsilon))^{-1/\epsilon}$.

Lemma 6: If $OPT \geq 1$ and $\beta = (|E|/(1 - \epsilon))^{-1/\epsilon}$, **MaxConcurrentFlow** terminates after at most $t = 1 + \frac{OPT}{\epsilon} \log_{1+\epsilon} \frac{|E|}{1-\epsilon}$ phases.

These lemmas require that $OPT \geq 1$. The running time of the algorithm also depends on OPT . Thus we need to ensure that OPT is at least one and not too large. Let ζ_i be the maximum flow value of commodity K_i when all other commodities have zero flow. Let $\zeta = \min_i \frac{\zeta_i}{\text{dem}(i)}$. Since at best all single commodity maximum flows can be routed simultaneously, ζ is an upper bound on f^* . On the other hand, routing $1/k$ fraction of each commodity flow of value ζ_i is a feasible solution, which implies that ζ/k is a lower bound on OPT . To ensure that $OPT \geq 1$, we can scale the original demands so that ζ/k is at least one. However, by doing so, OPT might be made as large as k , which is also undesirable.

To reduce the dependence on the number of phases on OPT , we follow the same technique adopted in [11] and [12]. If the algorithm does not stop after $T = \frac{2}{\epsilon} \log_{1+\epsilon} \frac{|E|}{1-\epsilon}$ phases, it means that $OPT > 2$. We then double demands of all commodities, so that OPT is halved and still at least 1. We then continue the algorithm, and double demands again if it does not stop after T phases.

Lemma 7: Given ζ_i for each commodity K_i , the running time of **MaxConcurrentFlow** is $O(\frac{\log |E|}{\epsilon^2} (2k \log k + |E|)) \cdot T_{mst}$.

Theorem 2: The total running time of **MaxConcurrentFlow** is $O(\frac{1}{\epsilon^2} [\log |E| (2k \log k + |E|) + k|E| (\log(|S_{max}| - 1) + \log U)]) \cdot T_{mst}$.

3.4 An Experiment (Part Two)

We conduct a simple experiment to illustrate how our algorithm works, based on the same setting as introduced in Sec. 3.2.

Table 4 shows the results of **MaxConcurrentFlow** with different approximation ratios. Here, we present the running time as the summary of two parts. The first part is the running time of the al-

Approximation Ratio	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
Rate of Session 1	131.77	131.85	132.07	132.27	132.46	132.58	132.78	132.89	133.05	133.20
Rate of Session 2	98.07	98.16	98.44	98.65	98.87	99.05	99.28	99.43	99.64	99.82
Overall Throughput	1182.89	1183.77	1186.17	1188.25	1190.25	1191.72	1193.80	1195.10	1196.87	1198.48
Number of Trees in Session 1	120	122	130	132	134	140	149	148	162	181
Number of Trees in Session 2	30	31	31	28	29	31	31	30	31	32
Running Time (number of MST operations)	1833+ 1376	2303+ 1703	2901+ 2135	3721+ 2794	5004+ 3767	7205+ 5389	11038+ 8393	19903+ 14935	44464+ 33292	176727+ 132672

Table 4: Experiment Results of MaxConcurrentFlow

gorithm shown in Table 3. As just discussed in the last subsection, the correctness and running time of **MaxConcurrentFlow** depends on some *a priori* knowledge of f^* . To acquire such knowledge, we first run **MaxFlow** algorithm for each session separately to obtain their maximum flow rates, then scale their demands such that $f^* \geq 1$ and is not too large. The overhead of this extra step is reflected in the second part.

From the data in Table 4, we have the same observation as from Table 2, except that the rate of session 2 is increased, at the price of dragging down the rate of session 1. The overall throughput also drops for the same reason. Note that although session 1 and 2 have the same demand, they are not necessarily required to have the same rate. The objective of **MaxConcurrentFlow** is to maximize the lower bound of any session’s rate, i.e., f^* . In other words, further lowering the rate of session 1 does not help increasing the rate of session 2. At this point, it is evident both analytically and experimentally that, the **MaxConcurrentFlow** algorithm achieves *weighted max-min* fairness, while the weights are identical to the demands of commodities $dem(i)$.

3.5 Discussions

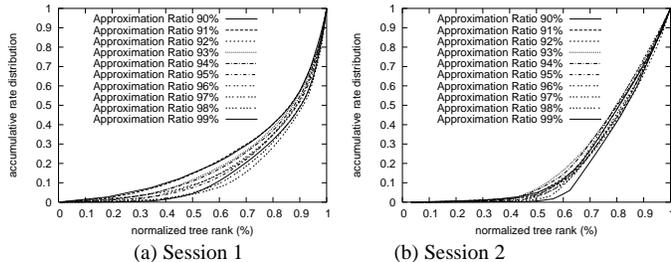


Figure 3: Overlay Tree Rate Distribution (MaxConcurrentFlow)

Fig. 3 plots the rate distribution among overlay trees returned by **MaxConcurrentFlow**. Here, we observe the same asymmetric rate distribution as in Fig. 2. This implies the possibility of more practical solutions, in which each session routes its commodity using a limited number of trees, but still approximates the optimal capacity utilization at a certain acceptable level. In the upcoming section, we will discuss the design and performance bounds of this type of algorithms.

In our experiment, all unicast paths of both overlay sessions cover 52 physical links. Fig. 4 plots the distribution of link utilization. It clearly shows that **MaxFlow** has stronger ability to utilize the link capacity than **MaxConcurrentFlow**. Also in both pictures, we observe a “stair case” phenomenon, i.e., the edges are grouped into different sets of distinct congestion levels. Our technical re-

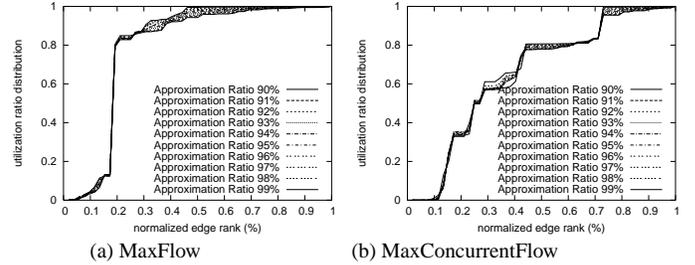


Figure 4: Link Utilization

port[13] reports our further study on this issue.

Finally, we note that the algorithms presented in this section are clearly not practical. First, they require that the commodity of each session can be arbitrarily split, which is not the case in practice. Second, too many overlay trees are required to support the derived session rate, even for small-sized sessions. However, with acceptable running time, they are able to infinitely approximate the theoretical optimal point of overlay capacity utilization, against which the performance of any practical solutions can be quantified. Therefore, we consider their major role as evaluation and benchmarking tools, which help us investigate the performance, applicability, and limitation of multi-tree overlay multicast solutions with fine granularity.

4. APPROXIMATING OPTIMAL CAPACITY UTILIZATION IN PRACTICAL SETTINGS

In this section, we consider the same problem in more practical settings. We first remove the assumption that each commodity can be split in arbitrary ways. Instead, it can only be decomposed into a finite number of sub-commodities, each with a specified demand. Second, each session only allows a limited number of trees in parallel, for the purpose of controlling management overhead.

4.1 Problem Formulation

In this paper, we focus on the maximum concurrent flow problem **M2**, for the purpose of achieving weighted max-min fairness. We add an integer variable x_i^j to formulate this problem. If each commodity K_i is unsplitable, i.e., it has to be routed along only

one overlay tree, then **M2** becomes

$$\begin{aligned}
\mathbf{M2_I} : \quad & \text{maximize} \quad f & (11) \\
\text{subject to} \quad & \sum_{j=1}^{|\mathcal{T}_i|} f_j^i \cdot x_j^i \geq f \cdot \text{dem}(i), i = 1, \dots, k \\
& \sum_{i=1}^k \sum_{j=1}^{|\mathcal{T}_i|} n_e(t_j^i) \cdot f_j^i \cdot x_j^i \leq c_e, \forall e \in E \\
& \sum_{j=1}^{|\mathcal{T}_i|} x_j^i = 1, i = 1, \dots, k \\
& f \geq 0, f_j^i \geq 0, x_j^i \in \{0, 1\}, \forall i, \forall j
\end{aligned}$$

M2_I is a 0 – 1 integer programming problem known as the *minimum congestion unsplittable flow* problem. It can be easily extended to the case when K_i has to be split into at most M trees. We can view K_i as M independent commodities which happen to have the same set of vertices. The sum of demands of these M commodities equals to $\text{dem}(i)$. Plus, each of them can only have one overlay tree.

Similarly, we can obtain the integer programming problem **P2_I** for **P2** as follows.

$$\begin{aligned}
\mathbf{P2_I} : \quad & \text{maximize} \quad f & (12) \\
\text{subject to} \quad & \sum_{j=1}^{\mathcal{P}_i} f_j^i \geq f \cdot \text{dem}(i), i = 1, \dots, k \\
& \sum_{i=1}^k \sum_{j=1}^{|\mathcal{P}_i|} n_e(p_j^i) \cdot f_j^i \cdot x_j^i \leq c_e, \forall e \in E \\
& \sum_{j=1}^{|\mathcal{P}_i|} x_j^i = 1, i = 1, \dots, k \\
& f \geq 0, f_j^i \geq 0, x_j^i \in \{0, 1\}, \forall i, \forall j
\end{aligned}$$

P2_I is NP-hard [14]. **M2_I** is also NP-hard, since **P2_I** is only a special case of **M2_I**, where all multicast sessions have only two members.

4.2 A Randomized Rounding Algorithm

If we replace $x_j^i \in \{0, 1\}$ with $x_j^i \in [0, 1]$, **M2_I** becomes **M2**, i.e., **M2** is the LP relaxation of **M2_I**. Let f be a feasible solution to **M2**, then $1/f$ can be understood as the maximum congestion over all $e \in E$, if we route $\text{dem}(i)$ units of traffic for each commodity K_i . Here, the congestion of e is defined as the ratio of total traffic routed through e and c_e . The objective of **M2** becomes to minimize the maximum congestion $1/f$. If f^* is the optimal objective value to **M2**, it is clear that the maximum congestion of any solution to **M2_I** is greater than $1/f^*$.

A popular approach to address an integer programming problem is randomized rounding [15]. In the case of **M2_I**, we first solve **M2**, then randomly choose an overlay tree for each commodity K_i , from the set of trees obtained from the solution to **M2**. The algorithm is listed in Table 5.

Here, l_e denotes the congestion of edge e . Each tree t^i is associated with an indicator l_{\max}^i to denote the maximum congestion along itself. l_{\max} is the maximum of all l_{\max}^i . Scaling $\text{dem}(i)$ by l_{\max}^i for each commodity K_i yields a feasible solution to **M2_I**. Let $OPT = 1/f^*$ be the optimal congestion, we have the following theorem.

Random-MinCongestion	
1	$\forall e \in E, l_e \leftarrow 0$
2	Solve M2 with MaxConcurrentFlow
3	for $i = 1$ to k do
4	Choose t_j^i with probability $\frac{f_j^i}{\sum_{j=1}^{ \mathcal{T}_i } f_j^i}$ as the overlay tree t^i for commodity K_i
5	$\forall e \in t^i, l_e \leftarrow l_e + \frac{n_e(t^i)\text{dem}(i)}{c_e}$
6	for $i = 1$ to k do
7	$l_{\max}^i \leftarrow \max_{e \in t^i} l_e$
8	$l_{\max} \leftarrow \max_{i=1}^k l_{\max}^i$

Table 5: Randomized Rounding

Theorem 3 Given $0 < \epsilon < 1$, if $OPT \geq 3 \ln(|E|/\epsilon)$, **Random-MinCongestion** returns a solution with maximum congestion $O(OPT + \sqrt{3 \cdot OPT \cdot \ln(|E|/\epsilon)})$, with probability at least $1 - \epsilon$.

Note that to obtain a small ϵ , OPT needs to be sufficiently large. This can be achieved by scaling up the demands of all commodities. However, doing so results in a worse approximation bound.

4.3 An Online Algorithm

The randomized rounding algorithm is of little practical value, as it needs to first work on the LP relaxation of our problem, then randomly select a subset of solutions and reroute all demands. We are interested to find a fast combinatorial algorithm, which has slightly worse approximation ratio, but routes the demand in only one iteration for each multicast session. Moreover, an online algorithm is desired, which can accept new sessions on the fly. In other words, upon the joining of a new session, the algorithm accumulatively adds routes for the new session, and only scales down the flow rate of existing sessions, instead of rerouting all sessions.

We extend Garg and Konemann’s scheme [11] to the domain of unsplittable flow problem, and propose an online algorithm with approximation ratio $\log(E)$, the best bound known for on-line algorithms so far [16]. The algorithm also works for **P2_I**, the source-destination unsplittable minimum congestion problem, with the same bound. It is listed in Table 6.

Online-MinCongestion	
1	$\forall e \in E, d_e \leftarrow \beta/c_e, l_e \leftarrow 0$
2	$f_j^i \leftarrow 0, t_j^i \in \mathcal{T}_i, i = 1, \dots, k$
3	for $i = 1$ to k do
4	$t^i \leftarrow$ minimum overlay spanning tree in \mathcal{T}_i using d_e
5	$f(t) \leftarrow f(t) + \text{dem}(i)$
6	$\forall e \in t^i, d_e \leftarrow d_e (1 + \rho \frac{n_e(t^i)\text{dem}(i)}{c_e})$,
7	$l_e \leftarrow l_e + \frac{n_e(t^i)\text{dem}(i)}{c_e}$
8	for $i = 1$ to k do
9	$l_{\max}^i \leftarrow \max_{e \in t^i} l_e$
10	$l_{\max} \leftarrow \max_{i=1}^k l_{\max}^i$

Table 6: An Online Algorithm

This algorithm continues to use the edge length assigning function introduced in Sec. 3. Here, ρ is the step size of the cost update. During iteration i , the algorithm finds the minimum overlay spanning tree t^i using the current edge length d_e , then routes $\text{dem}(i)$ units of traffic along t^i . The algorithm associates with each commodity K_i an indicator l_{\max}^i to denote its maximum congestion level. Finally, scaling $\text{dem}(i)$ by l_{\max}^i for each commodity K_i

returns a feasible solution. Let $OPT = 1/f^*$ be the optimal congestion, we have the following theorem.

Theorem 4 Online-MinCongestion returns a solution with maximum congestion $O(OPT \cdot \log |E|)$ if $OPT \geq 1/2$, and $O(2 \cdot OPT + \log |E|)$ otherwise.

To ensure the approximation ratio outlined in **Theorem 4** when f^* is unknown, we can scale down all demands to guarantee that $f^* \geq 2$. Recall that the “no-bottleneck” assumption can be achieved when $\frac{\max_{i=1}^k \text{dem}(i) \cdot |S_{\max}|}{\min_{e \in E} c_e} = 1$. This means that if there is only one session, it can be routed such that $f^* \geq 1$. Since the worst case happens when all k sessions are routed through one single link, we can ensure that $f^* \geq 2$ by letting $\frac{\max_{i=1}^k \text{dem}(i) \cdot |S_{\max}|}{\min_{e \in E} c_e} = 1/2k$.

4.4 An Experiment (Part Three)

We continue to use the same experiment setting in Sec. 3.2 and 3.4 to illustrate our algorithms. We test the performance of our algorithms by setting the limit on the number of trees from $n = 1$ to 20. For the random algorithm, we first run **MaxConcurrentFlow** with approximation ratio 95% to return a set of overlay trees that achieves optimal capacity utilization, then randomly choose n trees from the set, such that the probability a tree is selected is proportional to its contribution to the overall session rate. We repeat this procedure for 100 times, then report the average results. For the online algorithm, we replicate session 1 and 2 by $(n - 1)$ times, so that there are a total of $2n$ independent sessions. All these sessions have the same demand as 1. They join the network following a random sequence. We create 100 such sequences and report the average results.

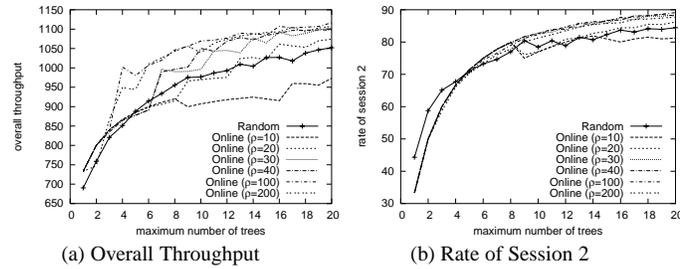


Figure 5: Throughput (Random and Online)

Fig. 5 shows that both algorithms greatly outperform their approximation lower bounds. In Fig. 5 (a), when $n = 20$, the overall throughput of the random algorithm exceeds 1000, more than 80% of the optimal throughput. It is even outperformed by the online algorithm, if we set $\rho \geq 30$. We also choose to show the rate of session 2 in Fig. 5 (b), since it reflects whether the algorithm is able to preserve fairness by minimizing the rate difference between session 1 and 2. Similar to Fig. 5 (a), when $n = 20$, both algorithms approximate the optimal objective value $f^* = 99.82$ derived in Sec. 3.4, by more than 80%. Also in both figures, we observe a clear trend of diminishing return of throughput growth as we increase the number of trees.

Fig. 6 shows the number of trees both algorithms actually return. Note that although we set a limit to the number of trees, say $n = 20$, in both algorithms, the same tree could be selected more than once. Comparing Fig. 6 with Fig. 5, we find out that the algorithm is able to achieve higher throughput as it diversifies its tree selection. This experiment shows that both algorithms efficiently utilize the

asymmetric rate distribution among overlay trees, as observed in Fig. 2 and 3, by selecting trees of higher rates.

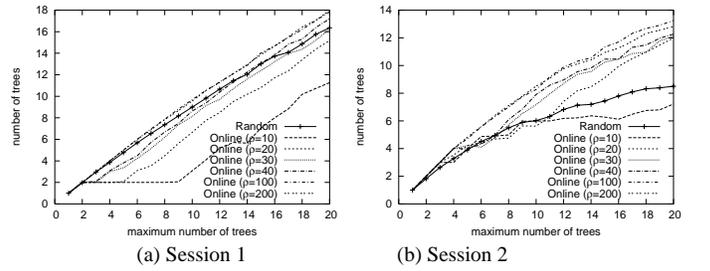


Figure 6: Number of Trees (Random and Online)

Finally, we discuss the role of ρ in the online algorithm. ρ controls the growing speed of the length of a physical link. Clearly, a large ρ rapidly increases a link e 's length when certain traffic is routed along e . This increases the probability that, when a new session joins, e will be unlikely to be included in the minimum overlay spanning tree, along which the traffic will be routed. In this way, other under-utilized links with smaller lengths will be selected.

In our proof of **Theorem 4**, in order to derive the desired approximation bound, we require that $\rho < f^*$. However, in this experiment, when we set $\rho = 200$, which is greater than $f^* = 99.82$, it does not hurt the performance of our algorithm, although it does not improve it either. At the time of this writing, we are unable to determine whether it is only a coincidence, or there exists alternative proofs which could remove such a condition. In fact, our algorithm already achieves the best performance when ρ is set to approximately equal to f^* . However, since f^* cannot be known a priori in practice, it is still preferable if ρ is not internally connected with f^* .

5. THE IMPACT OF IP ROUTING IN ACHIEVING OPTIMAL CAPACITY UTILIZATION

So far, we have explored several important issues concerning optimal capacity utilization in overlay network. The corresponding problems, as well as their solutions, have been presented. Another important issue we like to investigate is the role IP routing plays in all previous problems we have studied.

In Sec. 2, we define the overlay multicast tree as a tree spanning all members within a multicast session, where each tree link corresponds to the fixed IP route between its two end nodes. Obviously, such a routing strategy does not help to improve the capacity utilization of the physical network. Those “hot” links traversed by many IP routes can be easily saturated, while other ones selected by only few IP routes stay underutilized. Then our question is: how much is the impact of IP routing in deriving the achievable capacity utilization in overlay networks?

To answer this question, we first remove our previous assumption that any pair of nodes must route traffic via their pre-determined IP route. Instead, they can dynamically choose any unicast path between them. Then the overlay multicast tree has to be redefined as a tree spanning all members within a multicast session, where each tree link corresponds to an arbitrary unicast path between its two end nodes. Consequently, the previous studied problems **M1**, **M2**, **M2₁**, as well as their duals, have to be reformulated to accommodate such a redefinition. Note that the essence of algorithms (Table 1, 3, 5 and 6) to all these problems is to assign length d_e to each physical edge $e \in E$, such that the length of any overlay spanning tree, or the weighted sum of any overlay spanning tree from

each session, is at least 1. Thus, if we can find a way to calculate the minimum overlay spanning tree according to its new definition in polynomial time, then all previous algorithms can be applied to this new problem. Also, all previous theoretical conclusions hold (**Lemma 1 to 7, Theorem 1 to 4**), with the only change occurred to T_{mst} , the running time of minimum overlay spanning tree construction algorithm.

We now show how this algorithm works. For a multicast session S_i , we first construct an overlay graph for S_i , a complete graph $G_i = (S_i, E_i)$. Each edge $(v_m^i, v_n^i) \in E_i$ corresponds to the shortest unicast route between v_m^i and v_n^i based on the current length assignment to each physical edge $e \in E$. Then the length of (v_m^i, v_n^i) is the sum of lengths of all physical edges along this route. To calculate the lengths of all overlay edges in E_i , we can run the shortest path algorithm on each of them. Alternatively, using shortest path tree algorithm, we can get the lengths of all overlay edges from a given node (the root of the shortest path tree) to all other nodes. Thus, the running time of this operation is $|S_i|T_{spt}$, T_{spt} being the running time of the shortest path tree algorithm (The running time of Dijkstra’s algorithm is $|E| \log |V|$). After this step, we can obtain the minimum overlay spanning tree by running the minimum spanning tree algorithm on G_i . Therefore, the running time of the new algorithm exceeds the old algorithm by $|S_i|T_{spt}$, which is the overhead to calculate the length of each overlay edge in E_i .

Using the new algorithm, we are able to derive the optimal capacity utilization for a given group of overlay multicast sessions located within a given network, assuming arbitrary unicast routing. Comparing this result to the one obtained by the old algorithm which assumes IP unicast routing, we are able to quantify the impact of IP routing at constraining the optimal capacity utilization. If the difference between two results is significant, it means that IP routing is a major factor limiting the optimal performance of overlay multicast. Otherwise, it means that the impact of IP routing is minor, which implies that the major factors constraining the performance are the intrinsic properties of Internet, such as its topology.

6. RELATED WORK

Due to the difficulty of deployment of IP multicast, algorithms promoting application-layer overlay multicast have recently been proposed as remedial solutions, focusing on the issue of constructing and maintaining a multicast tree. The common objective is to perform multicast with only unicasts between end hosts, and to minimize the inefficiency brought forth by link stress and stretch. Narada [1], for example, constructs trees in a two-step process: it first constructs an efficient mesh among members, and in the second step construct a spanning tree of the mesh. More recently, researchers have focused on designing scalable overlay tree construction algorithms, using tools including Delaunay Triangulations [17] and organizing members into hierarchies of clusters [18].

Perhaps this work is more akin to two recent research papers that seek to utilize residual bandwidth availability by building multiple overlay multicast trees: CoopNet [19] and SplitStream [2]. CoopNet and SplitStream have proposed to utilize multiple multicast trees to deliver striped data, using either multiple description coding or source erasure codes. CoopNet proposes a centralized algorithm to facilitate using multiple multicast trees from different sources, and does not feature explicit built-in support of either maximizing capacity utilization, or achieving certain fairness. In contrast, SplitStream has proposed a decentralized algorithm to construct a forest of multicast trees from a single source. The main idea is to build multiple *interior-node-disjoint* trees, which guarantee that each node serves as internal forwarding nodes in only one

of the trees. SplitStream is developed based on Scribe, a tree-based multicast algorithm based on *structured* overlay networks.

This paper distinguishes from these previous work in many important aspects. Starting from problem formulations, our algorithms are designed from the ground up to evaluate the feasibility of constructing the best possible the data dissemination topologies beyond a single tree, and to achieve the optimum within certain approximation factors. We believe that the theoretical and experimental insights offered in this paper is important and noteworthy, since it provides guidance towards the design of realistic and distributed algorithms to optimize the performance of the constructed topologies. Towards this goal, our paper culminates in the proposal of an online approximation algorithm, which also meets the requirements of minimizing computation when new sessions are created. In addition, we consider the case where multiple concurrent sessions compete for overlay network capacities, where fairness constraints must be explicitly incorporated. In contrast, there are no provisions in both CoopNet and SplitStream regarding making such informed decisions with respect to topology construction, and existing algorithms are proposed based on intuitions rather than sound theoretical foundations. Similarly, none of the previous work has considered the impact of fairness on achieving optimized capacity utilization.

Finally, Kostic *et al.* [3] and Byers *et al.* [4] have both proposed to construct an overlay mesh of concurrent data dissemination connections, each sending a (hopefully) disjoint set of data. As a node receives data from these connections and merges incoming data, throughput may be significantly improved due to the larger number of concurrent connections. Byers *et al.* has discussed the algorithmic details of merging differences from different downloading sources, while Kostic *et al.* has proposed an elaborate algorithm that allows nodes to send data to different points in the overlay, as well as to locate and recover missing data items. Both work had similar objectives to ours, in the sense that they all seek to improve the bandwidth of data dissemination.

There are, at least, two significant differences comparing our work to these approaches. First, while both [3] and [4] need to assume large or unlimited buffers at each overlay node in order to store elements of data to potentially serve others, we do not make this assumption. While this assumption is certainly valid when file-based rather than in-memory buffers are used, it unavoidably lacks the support for *delay-sensitive* data dissemination, such as real-time streaming of multimedia or stock quotes. Second, our work shares the advantage of these approaches that the network capacity is as saturated as possible, without the complexity of locating missing data items from a potentially large number of possible hosts — data may *only* arrive from upstream nodes in the existing trees in the session.

7. CONCLUDING REMARKS

In this paper, we explore the entire spectrum of the multi-tree design philosophy when it comes to constructing data dissemination topologies in overlay networks. We first presents an array of combinatorial approximation algorithms to derive the optimal capacity utilization of overlay multicast when one or multiple competing sessions are present. Then we study a more realistic version of the same problem, where data flows can only be split into a specified number of subflows of fixed rates, and present a randomized-rounding and an online algorithm to address this problem. Finally, we revise all proposed algorithms to accommodate the case when arbitrary dynamic unicast routing, instead of fixed IP routing, is employed in the underlying physical network. By comparing results of two classes of algorithms in the same network setting, we

are able to quantify the impact of IP routing at constraining the optimal capacity utilization of overlay multicast.

Besides the simple experiment reported in this paper, we have conducted extensive simulations on synthetic and real Internet topologies, whose results can be found in [13]. Under our experimental settings, we have the following major findings. (1) Multi-tree solution can only utilize the capacity of the network by a small percentage (generally less than 50%), due to highly unbalanced link utilization ratio (recall Fig. 4). (2) Enforcing max-min fairness among competing sessions and maximizing overall throughput can be achieved simultaneously. (3) A simple online algorithm can largely approximate the upper bounds (more than 90%) of optimal throughput and optimal minimum session rate, with fairly small number of trees (around 20 to 30). (4) When repeating the same experiment without the restriction of IP routing, the performance results only improve by a small percentage generally less than 5%. This finding suggests that it might be the intrinsic property of current Internet topology that mainly constrain the performance of overlay multicast.

To the best of our knowledge, we are the first to discuss the important open problems of overlay fairness and capacity utilization, which may become a particularly practical and fertile area of research due to the exponentially increasing volume of active peer-to-peer data dissemination sessions being constructed in the Internet.

8. REFERENCES

- [1] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communications*, pp. 1456–1471, October 2002.
- [2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, October 2003.
- [3] D. Kotic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proc. of ACM SOSP*, 2003.
- [4] J. Byers and J. Considine, "Informed Content Delivery Across Adaptive Overlay Networks," in *Proc. of ACM SIGCOMM*, August 2002.
- [5] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, 1994.
- [6] M. Grotschel, L. Lovasz, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimizations*, Springer-Verlag, 1993.
- [7] W.T. Tutte, "On the problem of decomposing a graph in n connected factors," *Journal of London Mathematical Society*, vol. 36, pp. 221, 230, 1961.
- [8] C.S.J.A. Nash-Williams, "Edge-disjoint spanning trees of finite graphs," *Journal of London Mathematical Society*, vol. 36, pp. 445, 450, 1961.
- [9] W.H. Cunningham, "Optimal attack and reinforcement of a network," *Journal of ACM*, vol. 32, pp. 549, 561, 1985.
- [10] F. Barahona, "Packing spanning trees," *Mathematics of Operation Research*, vol. 20, pp. 104, 115, 1995.
- [11] N. Garg and J. Konemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," in *IEEE FOCS*, 1998.
- [12] L.K. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," *SIAM Journal of Discrete Mathematics*, vol. 13, pp. 505, 520, 2000.
- [13] Y. Cui, Baochun Li and K. Nahrstedt, "On achieving optimized capacity utilization in application overlay networks with multiple competing sessions," in *Technical Report UIUCDCS-R-2004-2425/UIIU-ENG-2004-1726*, 2004.
- [14] R.M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, pp. 85, 103, 1972.
- [15] P. Raghavan and C.D. Thompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, pp. 365, 374, 1987.
- [16] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts, "On-line routing of virtual circuits with applications to load balancing and machine scheduling," *Journal of ACM*, vol. 44, pp. 486, 504, 1997.
- [17] J. Liebeherr, M. Nahas, and W. Si, "Application-Layer Multicasting With Delaunay Triangulation Overlays," *IEEE Journal on Selected Areas in Communications*, pp. 1472–1488, 2002.
- [18] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. of ACM SIGCOMM*, August 2002.
- [19] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proc. of ACM NOSSDAV 2002*, May 2002.