

High-Bandwidth Routing in Dynamic Peer-to-Peer Streaming

Yi Cui
Department of EECS
Vanderbilt University
Nashville, TN
yi.cui@vanderbilt.edu

Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL
klara@cs.uiuc.edu

ABSTRACT

A fundamental problem in peer-to-peer streaming is building and maintaining high-bandwidth routing structure, which optimizes the receiver throughput. In this paper, we aim to design practical routing algorithms to this problem. The desired solution should accommodate the reality that nodes can frequently join and leave the overlay session, avoid global reorganization of the routing structure, let each node decide on its own how to attach to the existing tree, and maximally utilize the partial knowledge of the underlying physical network to optimize its performance. Based on these objectives, we design the dynamic high-bandwidth routing algorithm for peer-to-peer streaming. We prove the algorithm's approximation bound to the optimal rate. Experimental results show our algorithm to greatly outperform its theoretical bound at low management overhead and small number of multicast trees.

Categories and Subject Descriptors

C.2.5 [Local and Wide-Area Networks]: Internet (e.g., TCP/IP); D.4.8 [Performance]: Simulation; G.1.6 [Optimization]: Linear Programming; G.2.2 [Graph Theory]: Trees

General Terms

Algorithms, Measurement, Performance, Theory

Keywords

Overlay, Multicommodity Flow, Multicast, Online Routing

1. INTRODUCTION

Recently, overlay network has emerged as a powerful solution for a wide variety of content distribution applications. One of the most important applications is peer-to-peer streaming [1][2][3][4]. Other examples include bulk data distribution, live broadcasting, etc. Despite their diversified semantics, the common feature shared by these applications boils down to the overlay multicast routing problem: given a set of end nodes consisting of one sender and multiple receivers, how to effectively route the data from the sender to

all receivers. What distinguishes this problem from the traditional multicast is that each link in the overlay routing structure is not a physical link, but a unicast path connecting any pair of end nodes.

Depending on the optimization objective, the overlay multicast routing problem come in many flavors. In this paper, we consider the maximum throughput problem roughly presented as follows. We are given a set of overlay sessions coexisting on the same network. Each session consists of a sender and several receivers. How do we route the data via constructing multicast tree(s), such that the throughputs of these sessions are maximized in proportion to their demands?

This problem poses great practical values to many large-volume content distribution applications. Although challenging in its most basic setting, this problem is further complicated by many constraints from reality. For example, it is very common in peer-to-peer streaming that nodes frequently join and leave the session, which causes significant disturbance to the end-host-based routing structure. To name another difficulty, the complete knowledge of the underlying physical network cannot be available at the overlay layer, which often leads the overlay routing algorithm into suboptimal decisions.

Therefore, we are interested in finding a solution that can adapt to the above adversary conditions and can deliver *approximately* optimal performance. More specifically, we expect our solution to have the following features.

1. *Incremental Tree Construction*: the overlay tree must be incrementally built and dynamically updated as node joins or leaves the multicast.
2. *Localized Reorganization*: as dynamic change of membership can leave the tree into suboptimal shape, it needs to be reorganized from time to time. However, global reorganization is not desirable, and the frequency of reorganization must be limited.
3. *Distributed Management*: centralized solution is chosen over by the distributed scheme, where each node has partial view of the tree, and collaboratively maintains the tree with other nodes.
4. *Autonomous Decision*: each routing operation, may it be addition, deletion, or rearrangement of a tree edge, is triggered by the receiver node of this edge for its own benefit, requiring no permissions or negotiation with coexisting nodes.
5. *Partial Knowledge of Physical Topology*: the physical network is neither transparent nor invisible, but opaque to the overlay session. The routing algorithm should effectively utilize this partial knowledge to construct and maintain the approximately optimal overlay tree.

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, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

P2PMMS'05, November 11, 2005, Singapore.

Copyright 2005 ACM 1-59593-248-8/05/0011 ...\$5.00.

Based on the above design rationales, we propose a distributed routing algorithm to achieve maximum throughput in peer-to-peer streaming. The essential idea is to assign length to overlay edge as a function of its traffic load, and to use this length as the routing metric to help locating lightly-loaded overlay path, thus maintaining high-bandwidth multicast tree. We prove that the competitive ratio between the throughput returned by our algorithm and the optimal throughput is upper bounded in various node dynamic scenarios. Also these bounds remain unchanged when only partial knowledge of the physical topology is available to the algorithm. Our algorithm can be extended to the case of non-single-tree routing solutions. Experimental results show our algorithm to greatly outperform the proved approximation bound.

The rest of this paper is organized as follows. In Sec. 2, we formulate the problem in its most basic setting, which helps to illustrate the theoretical background of our routing algorithm. Sec. 3 presents our algorithm and its performance analysis. We first start from the single-tree case, the basic form of the algorithm, then proceed to modify it to accommodate the issue of topology unawareness and non-single-tree-based routing solutions. Sec. 4 presents the experimental results. We then discuss related works at Sec. 5 and conclude at Sec. 6.

2. MODEL

We formulate the routing problem in peer-to-peer streaming as a multicommodity flow problem. To illustrate the basic idea of our solution, we start from the simplest “static” setting, where a fixed number of multiple overlay sessions coexist in the network, and the members of each session are permanent.

2.1 Problem Formulation

Let $G = (\mathcal{V}, \mathcal{L})$ be a directed graph, with capacity c_l on each physical edge $l \in \mathcal{L}$. We are given a commodity set \mathcal{M} . Each commodity $m \in \mathcal{M}$ is an overlay session¹. Also, $S(m)$ is the sender, and $\mathcal{R}(m)$ is the set of receivers. $dem(m)$ is the demand of m , which is the desired flow rate from $S(m)$ to all nodes in $\mathcal{R}(m)$. We summarize these notations into the overlay graph model.

2.1.1 Overlay Graph

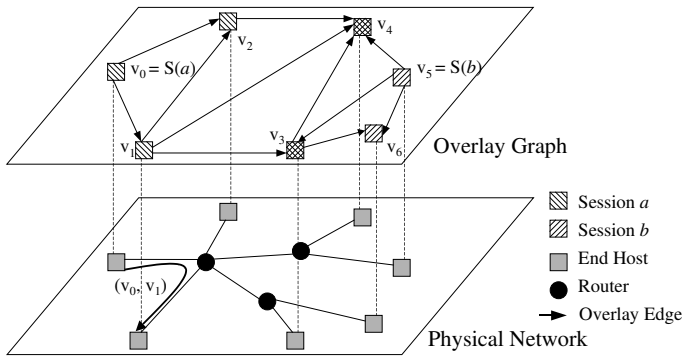


Figure 1: Illustration of Overlay Graph

For a session $m \in \mathcal{M}$, we define its overlay graph $G(m) = (\mathcal{V}(m), \mathcal{E}(m))$. In this graph, the node set $\mathcal{V}(m) = \{S(m)\} \cup \mathcal{R}(m)$ includes the sender and all receivers of session m . A directed edge $(v_s, v_t) \in \mathcal{E}(m)$ represents the data dependency between two nodes v_s and v_t , i.e., v_t can retrieve data from v_s . In

¹We will use the terms session and commodity interchangeably thereafter in this paper.

Fig. 1, two sessions, a and b , coexist in the same physical network. $\mathcal{V}(a) = \{v_0, v_1, v_2, v_3, v_4\}$. Here, node v_0 is the sender of session a , and nodes v_1 through v_4 are the receivers of a . For session b , $\mathcal{V}(b) = \{v_3, v_4, v_5, v_6\}$. Here, node v_5 is the sender of session b , and the rest belong to the receiver set $\mathcal{R}(b)$. Note that a node can belong to several sessions. In this example, v_3 and v_4 belong to both sessions a and b . Each overlay edge, e.g., (v_0, v_1) in Fig. 1, corresponds to the unicast route at the physical network.

2.1.2 Building Overlay Trees

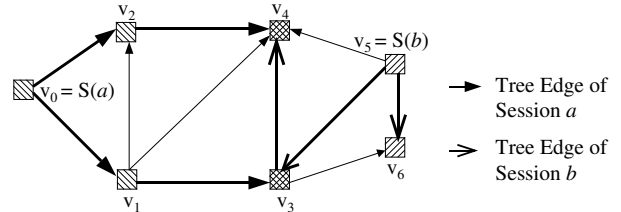


Figure 2: Illustration of Overlay Spanning Tree

On top of each overlay graph, different spanning trees can be found. Fig. 2 shows a sample spanning tree for the overlay graphs $G(a)$ and $G(b)$ in Fig. 1. With the formulation of overlay graph, the overlay routing problem boils down to, for each session $m \in \mathcal{M}$, finding a set of spanning trees on top of its overlay graph $G(m)$. The flow rate of session m is the aggregated rate of all its spanning trees. We collect these trees into the set $\mathcal{T}(m) = \{t_j(m)\}$. We use $f_j(m)$ to denote the flow of commodity m sent along the tree $t_j(m)$. Our goal is to maximize the flow rates of all sessions, formulated as the following linear programming problem.

M :

$$\text{maximize } f \quad (1)$$

$$\text{subject to } \sum_{j=1}^{|\mathcal{T}(m)|} f_j(m) \geq f \cdot dem(m), m \in \mathcal{M} \quad (2)$$

$$\sum_{m \in \mathcal{M}} \sum_{j=1}^{|\mathcal{T}(m)|} f_j(m) \cdot n_l(t_j(m)) \leq c_l, l \in \mathcal{L} \quad (3)$$

$$f \geq 0, f_j(m) \geq 0, 1 \leq j \leq |\mathcal{T}(m)|, \forall m \in \mathcal{M}$$

The objective of **M** is to maximize the scalar value f , subject to the fairness and capacity constraints. Inequality (2) enforces *fairness constraint* by requiring that the comparative ratio of traffic routed for different commodities satisfies the comparative ratio of their demands, i.e., at least $f \cdot dem(m)$ units of commodity flow can be routed simultaneously for each session $m \in \mathcal{M}$. Thus, the absolute value of $dem(m)$ is meaningless, as we can easily tune the value of f by scaling up/down all demands, while $f \cdot dem(m)$ stays unchanged. Inequality (3) specifies the *capacity constraint* that the traffic routed on each physical edge $l \in \mathcal{L}$ does not exceed its capacity c_l . Note here that $n_l(t_j(m))$ is an integer value denoting the number of appearances of l in $t_j(m)$. This value could be greater than one, since a physical edge may appear in a tree more than once, a unique feature of overlay network.

This problem, also known as “packing overlay spanning trees”[5], is proved to be solvable in polynomial time. This means we can derive the optimal value of the scalar f , denoted as f^* .

2.2 Solution Methodology

Before presenting the algorithm for **M**, we first formulate its dual as follows.

$$\begin{aligned}
 \mathbf{D} : \\
 \text{minimize} \quad & \sum_{l \in \mathcal{L}} c_l \cdot d_l \\
 \text{subject to} \quad & \sum_{l \in \mathcal{L}} n_l(t_j(m)) \cdot d_l \geq l(m), m \in \mathcal{M} \quad (4) \\
 & \sum_{m \in \mathcal{M}} z(m) \cdot dem(m) \geq 1 \quad (5) \\
 & d_l \geq 0, \forall l \in \mathcal{L}, z(m) \geq 0, \forall m \in \mathcal{M}
 \end{aligned}$$

D corresponds to the problem of assigning length d_l to each edge $l \in \mathcal{L}$ and weight $z(m)$ to each session $m \in \mathcal{M}$, such that for m , the length of any spanning tree in $\mathcal{T}(m)$ is at least $z(m)$, and the weighted sum of $z(m)$ by $dem(m)$ over all sessions is at least 1. By LP duality theory [6], the minimum of **D** is the maximum of **M**.

We can also interpret this problem from the viewpoint of supply and demand. Here, d_l represents the marginal price of l , i.e., the cost of using an additional unit of capacity of l . The more traffic is routed through l , the more expensive its bandwidth becomes, specified via d_l . In a similar fashion, $z(m)$ represents the marginal cost of not satisfying another unit of $dem(m)$, the traffic demand of session m .

Based on this interpretation, our algorithm is designed to run iteratively, in each iteration solving the primal **M** and dual **D** alternately.

1. *Initialization*: we assign initial length d_l for each physical edge $e \in \mathcal{L}$.
2. *Finding Minimum Overlay Spanning Tree*: for each session $m \in \mathcal{M}$, we first construct its *overlay graph*, a complete graph encompassing all members of m . Each edge corresponds to the unicast route connecting the two end nodes. The length of the edge is the aggregated length of all physical edges of this unicast route. Then we obtain the minimum overlay spanning tree by running the MST algorithm on top of the overlay graph.
3. *Routing Traffic*: on top of the found trees, we route different amounts of traffic in proportion to the demands of sessions they belong to.
4. *Edge Length Update*: for each edge $l \in \mathcal{L}$, we update its length based on its traffic increment.
5. *Repeat* steps 2) through 4), until the conditions (4) and (5) are met.

The essential idea of this algorithm is to always route traffic through the “cheapest route”, i.e., the most lightly-loaded overlay tree since the edge length can be understood as a function of its load. Interesting readers are referred to [5], which documented the detailed procedure of the algorithm, as well as definitions of the initial length d_l and the edge update function.

2.3 Discussion

The solvability of the problem **M** relies on the following unrealistic assumptions. If they are compromised, so will be the tractability of the problem.

1. *Permanent Membership*: each session is permanent and its membership does not change over time.
2. *Arbitrary Traffic Splitting*: the streaming traffic of each session can be arbitrarily split and fed into unlimited number of trees.
3. *Topology Awareness*: each overlay session has the complete knowledge of its underlying topology, as well as the capacity of each physical link.

If we remove the first assumption by allowing nodes to join or leave its overlay session on the fly, then in order to maintain the optimality, we need to rearrange trees for all sessions every time such an event happens. If we remove the second assumption by limiting the maximum number of trees each session can have, say k , it becomes the “ k -splittable” flow problem, which is NP-hard[7]. If the third assumption is changed, i.e., only partial knowledge of the physical network can be obtained from the overlay layer, the problem formulation does not hold any more.

Although new algorithms need to be designed to accommodate the realistic scenarios, the basic methodology presented in this section will survive and remain to play the central role. In what follows, we will progressively modify the above assumptions and shift towards new and more realistic settings. Meanwhile, we will present approximation algorithms, all of which return guaranteed performance bound to the optimal value f^* of the basic problem **M**.

Notation	Definition
$G = (\mathcal{V}, \mathcal{L})$	Physical Network
$m \in \mathcal{M}$	Overlay Session
$S(m)$	Sender of Session m
$\mathcal{R}(m)$	Receivers of Session m
$dem(m)$	Traffic Demand of Session m
$G(m) = (\mathcal{V}(m), \mathcal{E}(m))$	Overlay Graph of Session m
$(v_s, v_t) \in \mathcal{E}(m)$	Edge of Overlay Graph $G(m)$
$\mathcal{T}(m) = t_j(m)$	Multicast Tree of Session m
$f_j(m)$	Flow Rate of Tree $t_j(m)$
$l \in \mathcal{L}$	Physical Edge
c_l	Capacity of l
d_l	Edge Length of l
$n_l(t_j(m))$	Number of Times l Appears in $t_j(m)$
$z(m)$	Weight of Session m

Table 1: Notations used in Sec.2

3. ROUTING ALGORITHM

3.1 Overview

In this section, we present our overlay multicast routing algorithm to various application scenarios. We consider application scenarios along two dimensions: *node dynamics* and *data dependency*. We illustrate these concepts using the overlay graph model introduced in Sec. 2.

Along the dimension of *node dynamics*, we consider two cases. In the first case, nodes join the multicast session but never leave. Reflected in the overlay graph model, the graph is incremented gradually by the newly joined node. In the second case, nodes can join and leave a session anytime they want, i.e., the overlay graph can shrink or grow dynamically.

Along the dimension of *data dependency*, there are two types of dependency models. If the overlay graph is a directed acyclic

graph, there is a strong partial ordering of node dependency among nodes. Here, a node can only retrieve data from those which arrived earlier and already possessed the data, e.g., on-demand streaming. We referred to this model as *acyclic dependency*. The overlay graphs of session a and b in Fig. 1 exemplifies the acyclic dependency. In the second model, there is no restriction on from which node the data should be retrieved, which means the overlay graph can take any shape. We referred to this model as *arbitrary dependency*. This model is suitable for applications scenarios where data distribution is simultaneous, such as teleconferencing and live broadcasting.

The routing algorithm presented in this section works essentially the same way as the static algorithm introduced in Sec. 2.2: to find spanning tree(s) on the overlay graph of each session. However, the new algorithm is designed to accommodate the realistic condition that the overlay graph is constantly changed. The algorithm consists of three elementary functions. **Join** is run by each new node joining a multicast session. **Leave** is the operation for each old node leaving its session. **Rearrange** is performed by a present node in a session if it decides to improve its current tree edge by switching parent. As summarized in Tab. 2, these functions are combined differently to handle the various scenarios we consider.

	Acyclic Dependency	Arbitrary Dependency
Joining Only	Join	Join, Leave and Rearrange
Joining and Leaving	Join and Leave	Join, Leave and Rearrange

Table 2: Overview of Functions

In Sec. 3.2, we present these functions and theoretical results regarding their performance bounds. These functions are designed towards the single-tree structure, i.e., nodes within the same session are grouped into a single tree. In Sec. 3.3, we discuss how to modify our algorithm to accommodate the issue of topology unawareness, and how the algorithm can be extended for non-single-tree-based routing infrastructure, e.g., multiple trees or mesh.

3.2 Main Functions

We present the main functions of our algorithm. In this subsection, we target on the single tree case. Here, each session m maintains a tree $t(m) = (\mathcal{V}(m), \mathcal{E}(t(m)))$ connecting all nodes within the session.

3.2.1 Auxiliary Functions

We start by introducing some auxiliary functions designed to assist the main functions we are about to present. As previewed in Sec. 2, the essential idea of our algorithm is to assign lengths to physical edges, which indicate their current traffic conditions. The multicast tree operations are made based on the edge lengths as routing metrics. Therefore, the main usage of auxiliary functions is the initialization and update of edge lengths.

As shown in Tab. 3, for each physical edge $l \in \mathcal{L}$, two variables are allocated. d_l denotes its length, and σ_l denotes its current traffic load, i.e., the percentage of its occupied capacity. Also we define the length of an overlay edge e as the aggregate length of physical edges along its unicast route.

During the **Initialization**, d_l is initialized as a constant β normalized by its capacity c_l , and σ_l is set 0. The function **Load** is called when certain amount of traffic c is routed through an overlay edge e . In this function, for each physical edge l belonging to e , its length d_l and load σ_l are updated as shown in line 2. Finally, d_e is

Initialization

- 1 $\forall l \in \mathcal{L}, d_l \leftarrow \beta/c_l, \sigma_l \leftarrow 0$
- 2 $\mathcal{E}(t(m)) \leftarrow \phi$

Load(e, c) /* Load overlay edge e with traffic amount c */

- 1 $\forall l \in e$
- 2 $d_l \leftarrow d_l(1 + \rho \frac{c}{c_l}), \sigma_l \leftarrow \sigma_l + \frac{c}{c_l}$
- 3 $d_e \leftarrow \sum_{l \in e} d_l$

Unload(e, c) /* Unload overlay edge e with traffic amount c */

- 1 $\forall l \in e$
- 2 $d_l \leftarrow d_l/(1 + \rho \frac{c}{c_l}), \sigma_l \leftarrow \sigma_l - \frac{c}{c_l}$
- 3 $d_e \leftarrow \sum_{l \in e} d_l$

Table 3: Auxiliary Functions

updated accordingly. When certain traffic amount c along e is terminated, **Unload** is called, which basically reverses the operations of **Load**.

3.2.2 Join Function

When a new node v_{new} joins the session m , it runs the **Join** function to attach itself into the tree $t(m)$. According to the definition of the overlay graph $G(m)$ in Sec. 3.1, an overlay edge in $G(m)$ is directed from a node v to v_{new} if v can serve as the parent of v_{new} to retrieve data from. Therefore, all nodes which have an overlay edge directed towards v_{new} are its candidate parents. Among these nodes, our algorithm chooses the one with the minimum overlay edge length, i.e., the minimum-loaded overlay edge, as the parent v_{new} . Then we append this new edge into the tree $t(m)$ and update its edge length by calling the auxiliary function **Load**.

Join($v_{new}, t(m)$)

- 1 $p(v_{new}, t(m)) \leftarrow \operatorname{argmin}\{d(v, v_{new}) \mid (v, v_{new}) \in G(m)\}$
- 2 $\mathcal{R}(m) \leftarrow \mathcal{R}(m) \cup v_{new}$
- 3 $\mathcal{E}(t(m)) \leftarrow \mathcal{E}(t(m)) \cup (p(v_{new}, t(m)), v_{new})$
- 4 **Load**(($p(v_{new}, t(m)), v_{new}$), $dem(m)$)

Table 4: When Node v_{new} Joins Tree $t(m)$

The **Join** function alone suits best the (Joining Only)/(Acyclic Dependency) scenario, as outlined in Tab. 2. One example of this scenario is live broadcast. Here, nodes incrementally join the overlay session to watch certain live event, and the entire session is terminated when the broadcast is over. Also new nodes retrieve the content from those which joined earlier.

We measure the performance of our algorithm by comparing its achievable throughput $f_{achievable}$ by the optimal value f^* obtained via the ideal algorithm introduced in Sec. 2. More specifically, we define the *competitive ratio* of our algorithm as

$$\frac{f^*}{f_{achievable}}$$

and verify if there exists a worst case bound to the above ratio. We have the follow result.

Theorem 1: In the (Joining Only)/(Acyclic Dependency) scenario, the **Join** function alone returns the competitive ratio bounded by $\log(|\mathcal{L}|)$.

PROOF. **Part I:**

If there are m nodes which already joined the multicast, then what happened is a series consisting of k events. The i th event is a node $v^{(i)}$ joining a session $m^{(i)}$. $G^{(i)}(m)$ is the OG associated with $m^{(i)}$. $t^{(i)}(m)$ is the tree of $m^{(i)}$. Let $d_l^{(i)}$ denote the value of d_l after the i th event, we have

$$\begin{aligned} & \sum_{l \in \mathcal{L}} c_l \cdot d_l^{(i)} \\ &= \sum_{l \in \mathcal{L}} c_l \cdot d_l^{(i-1)} + \rho \cdot dem(m^{(i)}) \sum_{l \in (p(v^{(i)}, t^{(i)}(m)), v^{(i)})} c_l \frac{d_l^{(i-1)}}{c_l} \\ &= \sum_{l \in \mathcal{L}} c_l \cdot d_l^{(i-1)} + \rho \cdot dem(m^{(i)}) \cdot sd^{(i-1)}(v^{(i)}, t^{(i)}(m)) \\ &\leq \sum_{l \in \mathcal{L}} c_l \cdot d_l^{(i-1)} + \rho \cdot dem(m^{(i)}) \cdot sd^{(k)}(v^{(i)}, t^{(i)}(m)) \end{aligned}$$

where $sd^{(i-1)}(v^{(i)}, t^{(i)}(m))$ is the weight of the shortest overlay tree edge reaching $v^{(i)}$ after the $(i-1)$ th event, according to the **Join** algorithm. By the definition of the OG $G^{(i)}$, all candidate parents of $v^{(i)}$ already joined before $v^{(i)}$ does, and no more will join after that. Combining with the fact that the weight of each physical edge, hence the weight of each overlay edge, is non-decreasing, we obtain the last inequality. Therefore, after all k events, we have

$$\begin{aligned} & \sum_{l \in \mathcal{L}} c_l \cdot d_l^{(k)} \\ &\leq \sum_{l \in \mathcal{L}} c_l \cdot d_l^{(0)} + \rho \sum_{m \in \mathcal{M}} dem(m) \cdot \sum_{v \in \mathcal{R}(m)} sd^{(k)}(v, t^i(m)) \\ &= \beta |\mathcal{L}| + \rho \sum_{m \in \mathcal{M}} dem(m) \cdot z^{(k)}(m) \end{aligned} \quad (6)$$

By the formulation of problem $\sum_{v \in \mathcal{R}(m)} sd^{(k)}(v, t^i(m))$ is the aggregate weight of the minimum spanning tree for session m_i based on the edge weight after all k events.

Part II: Problem **D** tries to minimize $\sum_{l \in \mathcal{L}} c_l \cdot d_l$ subject to the constraint that $\sum_{m \in \mathcal{M}} dem(m) \cdot z(m) \geq 1$. Alternatively, this objective can be reformulated as follows.

$$\mathbf{D2}' : \text{ minimize } \frac{\sum_{l \in \mathcal{L}} c_l \cdot d_l}{\sum_{i=1}^k dem(m_i) \cdot z_i} \\ d_l \geq 0, \forall l \in \mathcal{L}, z_i \geq 0, i = 1, \dots, k$$

Also because the optimums of problems **M2** and **D2'** are equal, from (6), we have the following inequality if $\rho < f^*$.

$$\frac{\beta |\mathcal{L}|}{1 - \rho/f^*} \geq \sum_{l \in \mathcal{L}} c_l \cdot d_l^{(m)} \geq c_{l_{\max}} \cdot d_{l_{\max}}^{(m)} \quad (7)$$

where e_{\max} is the edge whose load is the maximum among all edges in \mathcal{L} . We denote its load as σ_{\max} . By the definition of edge update function,

$$\frac{c_{l_{\max}} d_{l_{\max}}^{(m)}}{\beta} = \prod_{i=1, \dots, m}^{(p(v^{(i)}, t^{(i)}), v^{(i)}) \text{ goes through } e_{\max}} \left(1 + \rho \frac{dem(m^{(i)})}{c_{l_{\max}}}\right) \quad (8)$$

Since $(1 + x\rho) \geq (1 + \rho)^x$ when $x \leq 1$,

$$(8) \geq (1 + \rho)^{\sum_{i=1, \dots, m}^{(p(v^{(i)}, t^{(i)}), v^{(i)}) \text{ goes through } e_{\max}} \frac{dem(m^{(i)})}{c_{l_{\max}}}} = (1 + \rho)^{\sigma_{\max}}$$

if $\frac{dem(m^{(i)})}{c_{l_{\max}}}$ is always no greater than 1, i.e., the traffic routed during each iteration does not overflow e_{\max} . We refer to this requirement as the “no-bottleneck” assumption, which can be achieved by scaling all demands such that $\frac{\max_{m \in \mathcal{M}} dem(m)}{\min_{l \in \mathcal{L}} c_l} = 1$. Coming back to (7), we have

$$(1 + \rho)^{\sigma_{\max}} \leq \frac{|\mathcal{L}|}{1 - \rho/f^*}$$

which implies

$$\begin{aligned} \sigma_{\max} &\leq \log_{1+\rho} \frac{|\mathcal{L}|}{1 - \rho/f^*} = \frac{\log |\mathcal{L}| - \log(1 - \rho/f^*)}{\log(1 + \rho)} \\ &\leq \frac{\log |\mathcal{L}| - \log(1 - \rho/f^*)}{\rho} \end{aligned} \quad (9)$$

The last inequality of (9) holds when $\rho \leq 1$. Also since it must follow that $\rho < f^*$, ρ can be determined as follows.

If $f^* < 2$, we can set $\rho < 1$, and satisfy that ρ/f^* is any constant factor less than $1/2$. This makes $\log(1 - \rho/f^*)$ a constant too. Then (9) becomes

$$\sigma_{\max} \leq O\left(\frac{\log |\mathcal{L}|}{f^*}\right)$$

If $f^* \geq 2$, we can set $\rho = 1$. Then (9) becomes

$$\sigma_{\max} \leq \log |\mathcal{L}| - \log(1 - 1/f^*) < \log |\mathcal{L}| + \frac{2}{f^*}$$

Since $OPT_{cong} = 1/f^*$ is the optimal congestion, we complete the proof. \square

3.2.3 Leave Function

If nodes are allowed to leave the multicast tree $t(m)$, then **Leave** function needs to be called. In this function, the leaving node v_{old} first terminates the incoming traffic from its parent, then stops feeding its children, if there are any. Finally, the auxiliary function **Unload** is called to update the lengths of corresponding overlay edges. Note that the orphaned children of v_{old} need to find themselves new parents by rejoining the session m as a new node and calling the function **Load**.

Leave ($v_{old}, t(m)$)	
1	$\mathcal{R}(m) \leftarrow \mathcal{R}(m) - v_{old}$
2	$\mathcal{E}(t(m)) \leftarrow \mathcal{E}(t(m)) - (p(v_{old}, t(m)), v_{old})$
3	Unload (($p(v_{old}, t(m)), v_{old}$), $dem(m)$)
4	$\forall v_{child} \in \{v_{child} \mid (v_{old}, v_{child}) \in \mathcal{E}(t(m))\}$
5	$\mathcal{E}(t(m)) \leftarrow \mathcal{E}(t(m)) - (v_{old}, v_{child})$
6	Unload ((v_{old}, v_{child}), $dem(m)$)
7	Join (v_{child}, m)

Table 5: When Node v_{old} Leaves Tree $t(m)$

The **Leave** function, combined with the **Join** function, can handle the (Joining and Leaving)/(Acyclic Dependency) scenario, such as asynchronous Video-on-Demand. Here, new nodes join the session by attaching to the earlier nodes which already acquired the partial content. However, each node will stay online for a finite amount of time. It will leave the multicast once it receives the entire content. In this scenario, we have the following result.

Theorem 2: In the (Joining and Leaving)/(Acyclic Dependency) scenario, the combination of **Join** and **Leave** functions returns the competitive ratio bounded by $\log(|\mathcal{L}|T)$, T being the ratio of the maximum online duration and the minimum one.

3.2.4 Rearrange Function

The **Rearrange** function is designed for present nodes in a multicast session to improve its incoming throughput by switching parent. When the node v_{pres} joined the multicast by running the **Join** algorithm, it chooses to attach to the existing multicast tree via the overlay edge with the minimum length. However, due to the dynamics of node joining and leaving, the tree edge of v_{pres} cannot always guarantee to have the minimum length. Therefore, it may be better for v_{pres} to periodically switch to a new parent whose overlay edge is currently the shortest. However, certain tradeoff needs to be maintained to prevent such rearrangement from happening too frequently. We introduce a threshold value α , and require that v_{pres} switches parent only when the ratio between the length of its existing tree edge and the length of the shortest overlay edge exceeds α .

<pre> Rearrange($v_{pres}, t(m)$) 1 $v_{min} \leftarrow \operatorname{argmin}\{d(v, v_{pres}) \mid (v, v_{pres}) \in G(m)\}$ 2 If $d(p(v_{pres}, t(m)), v_{pres}) > \alpha \cdot d(v_{min}, v_{pres})$ 3 $\mathcal{E}(t(m)) \leftarrow \mathcal{E}(t(m)) \cup (v_{min}, v_{pres}) -$ $(p(v_{pres}, t(m)), v_{pres})$ 4 Unload($(p(v_{pres}, t(m)), v_{pres}), dem(m)$) 5 Load($(v_{min}, v_{pres}), dem(m)$) 6 $p(v_{pres}, t(m)) \leftarrow v_{min}$ </pre>

Table 6: Rearrangement of Node v_{pres} in Tree $t(m)$

The **Rearrange** function, combined with the **Join** and **Leave** function, are designed for the (Joining and Leaving)/(Arbitrary Dependency) scenario, mainly targeted to interactive applications such as video conferencing and instant messaging. In this applications, nodes are free to join and leave the multicast there at anytime, and there is no strong sequential order regarding the data relay among participants in multicast. We show that if parent-switching rearrangement is allowed, our algorithm can achieve the following performance bound.

Theorem 3: In the (Joining and Leaving)/(Arbitrary Dependency) scenario, the combination of **Join**, **Leave** and **Rearrange** functions returns the competitive ratio bounded by $\log(|\mathcal{L}|)$, and each node will go through at most $\log_\alpha(|\mathcal{L}|)$ rearrangements.

3.3 Extensions

3.3.1 Partial Knowledge of Physical Topology

The algorithm presented in Sec. 3.2 assumes that each overlay session has entire knowledge of its underlying physical network. This means that in order to function properly, the algorithm must know the complete topology of the underlying network, and the capacity of each physical link.

However, such complete knowledge can be hardly acquired in practice, if not at all impossible. Based on this concern, we modify our algorithm assuming that only partial knowledge of the underlying network is available. In the modified algorithm, we choose to let the main functions stay intact. Instead, the link weight update function is redefined from physical-link-based to overlay-edge-based. Therefore, we only need to update the original auxiliary functions (Tab. 3) to the new ones shown in Tab. 7.

Initialization

```

1   $\forall l \in \mathcal{L}, \sigma_l \leftarrow 0$ 
2   $\forall e \in \mathcal{E}(m), d_e = |e|\beta, c_e = \min\{c_l \mid l \in e\}$ 
3   $\mathcal{E}(t(m)) \leftarrow \phi$ 

```

Load(e, c) /* Load overlay edge e with traffic amount c */

```

1   $d_e \leftarrow d_e(1 + \rho \frac{c}{c_e})$ 
2   $\forall l \in e$ 
3   $\sigma_l \leftarrow \sigma_l + \frac{c}{c_l}$ 

```

Unload(e, c) /* Unload overlay edge e with traffic amount c */

```

1   $d_e \leftarrow d_e / (1 + \rho \frac{c}{c_e})$ 
2   $\forall l \in e$ 
3   $\sigma_l \leftarrow \sigma_l - \frac{c}{c_l}$ 

```

Table 7: Modified Auxiliary Functions

/* When node v_{new} Joins Session m */

```

Join( $v_{new}, m$ )
1  for  $j = 1$  to  $T(m)$ 
2  Join( $v_{new}, t_j(m)$ )

```

/* When node v_{old} Leaves Session m */

```

Leave( $v_{old}, m$ )
1  for  $j = 1$  to  $T_i$ 
2  Leave( $v_{old}, t_j(m)$ )

```

/* Rearrangement of node v_{pres} in Session m */

```

Rearrange( $v_{pres}, m$ )
1  for  $j = 1$  to  $T(m)$ 
2  Rearrange( $v_{pres}, t_j(m)$ )

```

Table 8: Modified Algorithm for the Multi-Tree Case

In the modified auxiliary functions, the weight of an overlay edge e depends on the following information: (1) $|e|$, the number of physical links contained in the unicast route of e , and (2) c_e , the capacity of the bottleneck physical link on this route. The weight of e is initialized as a constant β timed by the length $|e|$, and normalized by its capacity c_e . This gives smaller weights to those overlay edges with shorter unicast route and larger bottleneck capacity, which are in turn more likely to be chosen by the tree construction algorithm. The weight update functions in **Load** and **Unload** are identical with their original versions, where an overlay edge is treated the same way as a physical link.

In our technical report[?], we prove that, in conjunction with the modified auxiliary functions, results in **Theorem 1** through **3** also hold.

3.3.2 Beyond the Single Tree Structure

So far, we have targeted on the single tree case, i.e., only one tree is built and maintained for each overlay session. Our algorithm can be easily upgraded to the multi-tree case. The modified algorithm is shown in Tab. 8, where each session m can have $T(m)$ trees.

In the modified algorithm, when joining the session m , each node independently joins each tree of the session the same way as in the original algorithm. In other words, from the algorithm's viewpoint, a session with $T(m)$ trees is no different to $T(m)$ single-tree sessions with identical member set and traffic demand. Therefore, **Theorem 1** through **3**, as well as their counterparts in the case of modified auxiliary functions, still hold in the multi-tree case.

3.4 Practical Issues

3.4.1 Overlay Graph Management

Our tree construction algorithm runs on top of the overlay graph. For example, upon joining a overlay session, the new node needs to first know which nodes are its neighbors in the OG, then choose the one with the shortest distance in terms of overlay edge weight. Therefore, certain bootstrapping mechanism has to be in place to help the new nodes acquire such information.

OG can be managed in the centralized, where the server of each session maintains the entire OG by keeping track of all its receivers. Each receiver should notify the server upon joining and leaving the session. Heartbeat messages are also necessary to be exchanged among the server and receivers in case any receiver node leaves without notice or suddenly crashes.

3.4.2 Physical Network Measurement

Besides the OG neighbor list, each node must measure the weight of overlay edges connecting to these neighbors. As stated in Sec. 3.3, in order to compute the weight of an overlay edge, we need to know the hop count and bottleneck capacity of its unicast route. Such information can be obtained using the following techniques. The number of physical links (hop count) can be found by network path finding tools such as traceroute. The bottleneck bandwidth along the unicast route of an overlay edge can be measured by tools such as Packet Pair[8], pathrate[9], in an end-to-end manner.

Notation	Definition
$t(m) = (\mathcal{V}(m), \mathcal{E}(t(m)))$	Overlay Tree of Session m
$\mathcal{E}(t(m))$	Overlay Edge Set of Tree $t(m)$
$p(v, t(m))$	Parent Node of v in Tree m
$T(m)$	Number of Trees of Session m
$d(v_s, v_t)$	Edge Length of Overlay Edge (v_s, v_t)
$e \in \mathcal{E}(m)$	Edge of Overlay Graph $G(m)$
c_e	Capacity of Overlay Edge e
d_e	Length of Overlay Edge e
$ e $	Number of Physical Edges e Contains
β	Initial Value of Physical Edge d_i
ρ	Step Size

Table 9: Notations used in Sec.3

4. EXPERIMENTS

4.1 Experimental Setup

In this section, we experimentally evaluate the performance of our algorithm. We setup two application scenarios. Due to page limit, we only report one of the application scenarios we have experimented with, where we use the Boston BRITE topology generator to create a 100-node router-level network using the Waxman model. The bandwidth of each network edge is uniformly distributed from 10Mbps to 1024Mbps. On top of this network, we create two overlay sessions. We employ the acyclic dependency model where the OG for each session is a DAG, i.e., each node can only attach to those which joined the session earlier. A total of 50 nodes within the network randomly choose to join either session following the Poisson process. The average inter-arrival time is 30 seconds. Each node stays online for a duration exponentially distributed from 1 minute to 1 hour.

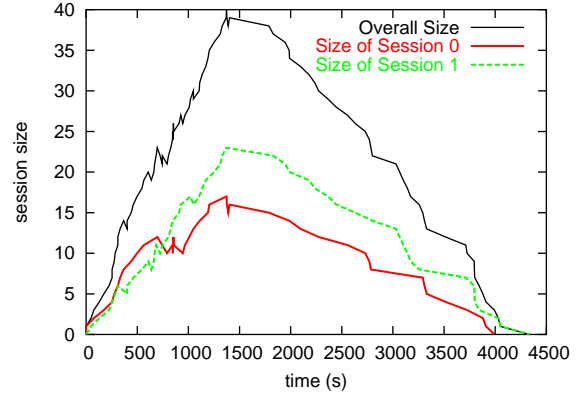


Figure 3: Session Size

4.2 Rate Comparison

We first study the performance of our algorithm in the simplest case. Here, each session only maintains a single tree. Each node, once joined the session, does not rearrange its own position in the tree unless orphaned by its parent.

We first compares the performance of our algorithm in Scenario A with the optimal algorithm[5]. As shown in Fig. 4, the rate returned by our algorithm is at least half of the maximum rate returned by the optimal algorithm. However, in order to achieve the optimal rate, a large number of trees are needed. As illustrated in Fig. 5, when the session size reaches over 10, the number of trees returned by the optimal algorithm exceeds 100000. Seemingly surprising, this number is only a very small percentage of all possible trees. According to the Cayley's theorem, for a multicast session of n receivers, there are $(n + 1)^{n-1}$ different trees. Therefore, as the number of trees grows exponentially when the session size increases (Fig. 5 (a)), its percentage to the number of all possible trees actually declines (Fig. 5 (b)).

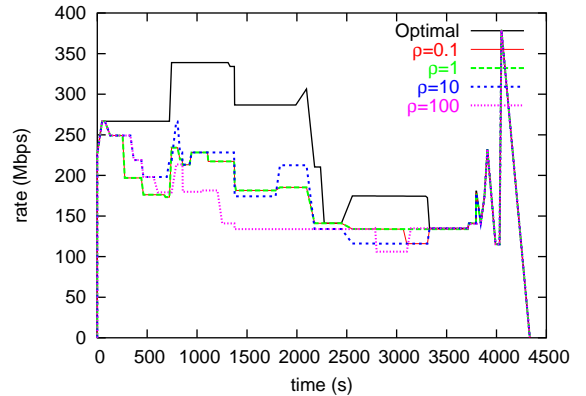


Figure 4: Average Rate (Single Tree, No Rearrangements)

Besides, the value of ρ plays an important role in the performance of our algorithm. As the stepsize of edge weight update function, ρ decides how sensitive the weight update function reacts to traffic update. If ρ is too small, then the edge weight may increase too slowly, causing the already loaded edges still chosen as the shortest route. If ρ is too large, the weight of an edge can be incremented dramatically, exaggerating its traffic condition. In either case, the algorithm will not be able to maintain trees that

maximally utilize the network bandwidth since the traffic condition is not correctly reflected through the edge weight. In Fig. 4, we set the value of ρ from 0.1 to 100, which shows our algorithm to return the best performance when $1 \leq \rho \leq 10$.

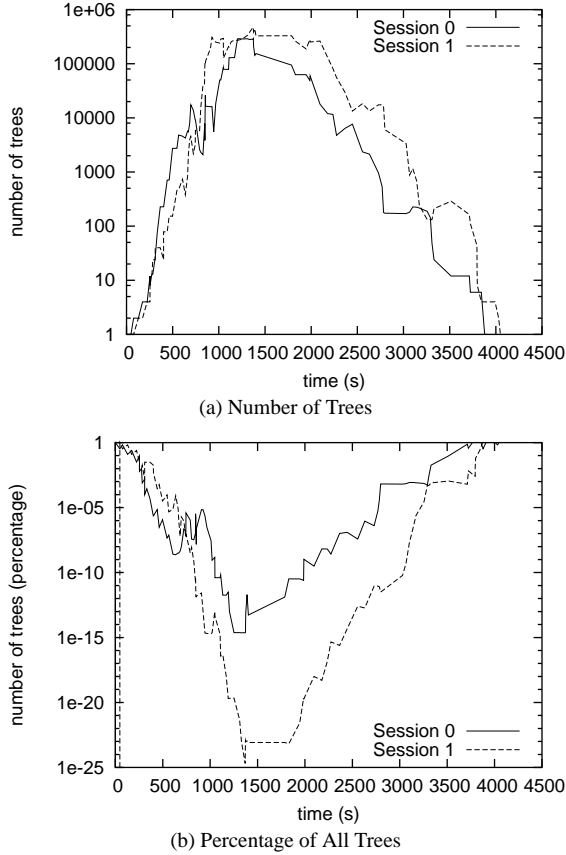


Figure 5: Number of Trees Required to Achieve the Optimal Rates

4.3 Effects of Multiple Trees

We further testify how building multiple trees helps improve the performance. In Fig. 6 (a), when $\rho = 1$, the performance gain is considerable when two trees, instead of one tree, are made for each session. However, the diminishing return is also obvious when we further increase the number of trees. As reminded in Fig. 4, the rate of the single-tree solution already matches at least half the rate returned by the optimal algorithm, which sets the upper bound for the performance of any algorithm. Also shown from the same figure, setting too many trees (16 trees) actually negatively impacts the algorithm performance, causing the returned rate to be lower. In Fig. 6 (b), we shrink the value of ρ to 0.1, and find out that the algorithm performs the best when there are 16 trees. On the other hand, the rate improves trivially in the case of two trees. In summary, the smaller ρ is, the more trees we need to achieve the best performance. When the number of trees is over this limit, the performance starts to degrade.

4.4 Effects of Rearrangement

Fig. 4.4 quantifies how dynamic tree rearrangements help improve the performance. Here, the value of α determines the threshold for a node to switch its parent, i.e., it will switch to an overlay edge whose weight is α times smaller than its current tree edge. In

case the dependency model is arbitrary, enforcing rearrangement can improve the average rate significantly. This is because for an existing node v_{pres} in the tree, a new-coming node may become its candidate parent and have a small-weighted (i.e., lightly-loaded) overlay edge to v_{pres} , causing it to switch parent. On the other hand, in case of acyclic dependency (Fig. 4.4), a node arriving later than v_{pres} can never become its candidate parent. This means that the candidate parent list of v_{pres} stays unchanged from the moment it joins the session, which renders the effect of rearrangements very limited.

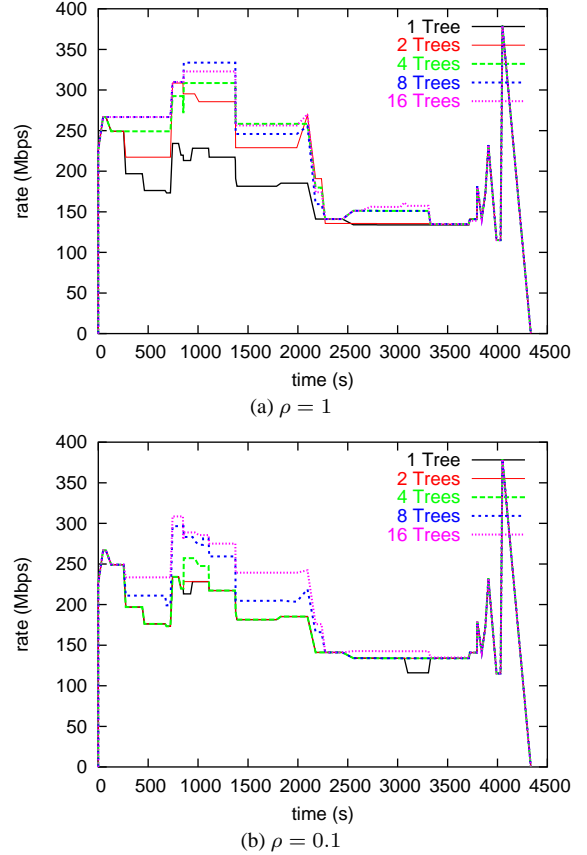


Figure 6: Effects of Multiple Trees (Scenario A, No Rearrangements)

Also we see that setting smaller α can occasionally improve the performance further. However, the overhead is significant. Fig. 7 records the total number of rearrangements throughout the sequence. As a reference, we also record the number of reconnections due to parent leaving, which is the mandatory overhead. It is shown that linearly lowering the value of α exponentially increase the frequency of rearrangements. Therefore, enforcing the rearrangement with high switching threshold value seems to be a good option in case of the arbitrary dependency OG.

4.5 Effects of Overlay-based Weight Update Functions

Now we replace our algorithm with the overlay-based weight update functions introduced in Sec. 3.3, and repeat the above experiments to compare the performance of different weight update functions. Fig. 8 corresponds to Fig. 4. Fig. 9 corresponds to Fig. 6. Fig. 10 corresponds to Fig. 4.4. Fig. 11 corresponds to Fig. 7.

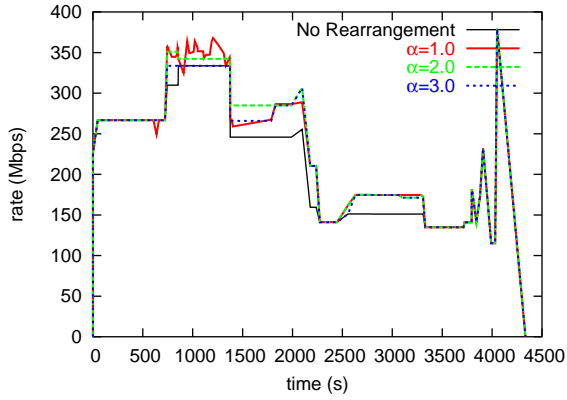


Figure 8: Average Rate with Overlay-based Weight Update Functions (2 Trees, No Rearrangements)

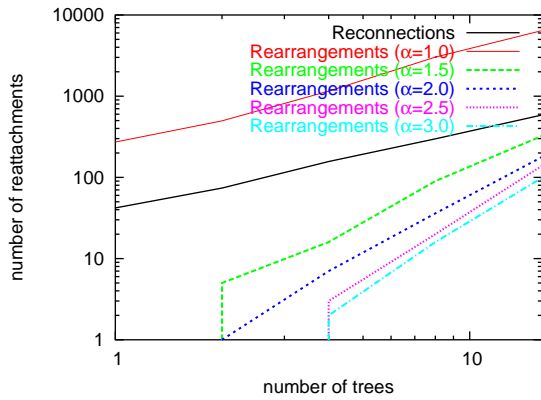


Figure 7: Number of Rearrangements ($\rho = 1$)

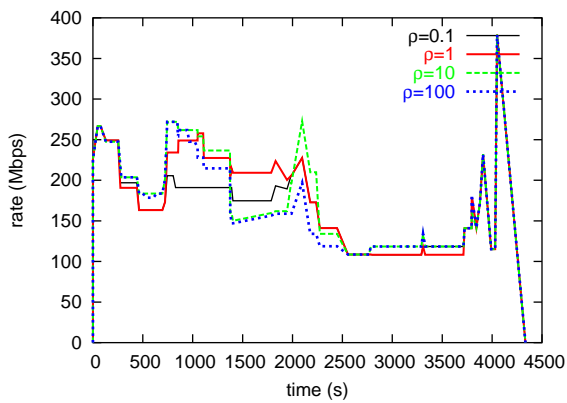


Figure 11: Number of Rearrangements with Overlay-based Weight Update Functions ($\rho = 1$)

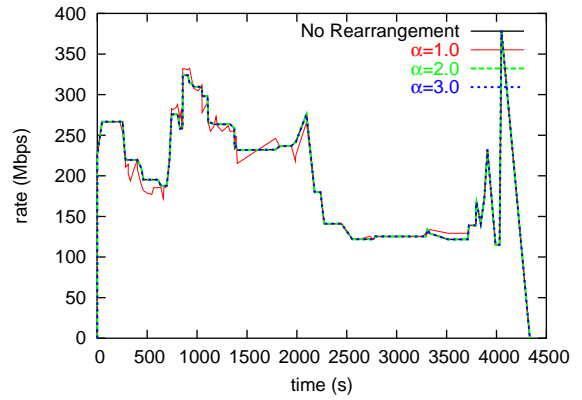


Figure 10: Effects of Rearrangements with Overlay-based Weight Update Functions (8 trees, $\rho = 1$)

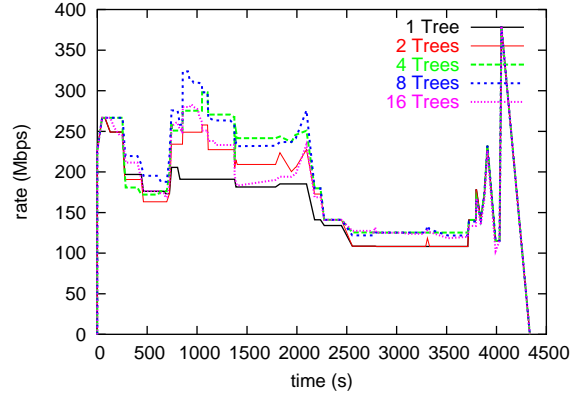


Figure 9: Effects of Multiple Trees with Overlay-based Weight Update Functions ($\rho = 1$)

captionEffects of Rearrangements (8 trees, $\rho = 1$)

The results show the performance of algorithm with overlay-based weight update function to be comparable with the original algorithm using weight update functions for physical links. This means that our algorithm is able to fully utilize the partial knowledge of physical network and return performance far exceeding its theoretical upper bound.

5. RELATED WORK

The idea of assigning weights to links to reflect their traffic conditions has been extensively explored in the domain of online unicast routing [10] [11]. The same idea has also been extended to the multicast domain, in case receivers within a multicast session arrive in batch [12] or separately [13]. In this work, we further extend the idea to the domain of overlay network. Furthermore, we use a new weight update function different to the exponential function adopted by these solutions. A unique challenge of this communication paradigm is that multicasting and data routing are carried out at separate network layers. Hence, overlay nodes cannot obtain the entire knowledge or control over the underlying physical network. Our algorithm proves that, with partial knowledge of the physical network (hop count and bottleneck capacity of the overlay edge), the same performance bound in the case of unicast routing can still be achieved.

Several works propose overlay tree/network construction solutions that take into account the topology of the physical network. In TAG (Topology-Aware Grouping) [14], the information about overlap in routes to the sender among group members is used to guide the construction of overlay tree. The resulting tree has low relative delay penalty, and introduces a limited number of identical packets on the same link. In [15], a distributed binning scheme is proposed where overlay nodes partition themselves into bins such that nodes that fall within a given bin are relatively close to one another in terms of network latency. A landmark node is assigned for each bin, and each node determines which bin it belongs to by measuring its distance to these landmark nodes. Since the location of landmark nodes are well-known, each overlay node can infer its distance to other nodes, which in turn aids the construction of the overlay graph.

Focusing on minimizing data latency and packet duplication, both works use latency as the main metric. On the other hand, since the goal of our work is to maximize the throughput of all multicast sessions while maintaining fairness, we use the available link bandwidth as the main metric used in our overlay edge weight update function, which helps build the tree with higher throughput. However, our solution can easily adopt the latency criterion by imposing the delay bound into the selection of high-throughput route.

Finally, [16] shares the same goal with our work: to maximize the overlay throughput by exploring the route diversity of overlay network. The proposed solution is to build multiple disjoint overlay MSTs for each session. However, since the routing metric is defined as the static latency, not the dynamic weight to reflect the traffic condition, this solution does not have any performance guarantee.

6. CONCLUSIONS

In this paper, we aim to find practical routing solutions for maximum-throughput peer-to-peer streaming. The desired solution should accommodate the reality that nodes can frequently join and leave the overlay session, avoid global reorganization of the routing structure, let each node decide on its own how to attach to the existing tree, and maximally utilize the partial knowledge of the underlying

physical network to optimize its performance. Based on these objectives, we design the dynamic high-bandwidth routing algorithm for peer-to-peer streaming. We prove the algorithm's approximation bound to the optimal rate. Experimental results show our algorithm to greatly outperform its theoretical bound at low management overhead and small number of multicast trees.

7. REFERENCES

- [1] D. Xu, M. Hefeeda, S. Hambruch, and B. Bhargava, "On peer-to-peer media streaming," in *IEEE International Conference on Distributed Computing Systems*, 2002.
- [2] M. Hefeeda, D. Xu, A. Habib, B. Bhargava, and Boyan Botev, "Collectcast: A peer-to-peer service for media streaming," *ACM Multimedia Systems Journal*, 2005.
- [3] X. Zhang, J. Liu and B. Li, and T.-S. P. Yum, "Donet/coolstreaming: A data-driven overlay network for live media streaming," in *IEEE INFOCOM*, 2005.
- [4] Y. Cui and K. Nahrstedt, "Layered peer-to-peer streaming," in *Proc. of ACM NOSSDAV*, 2003.
- [5] Y. Cui, B. Li, and K. Nahrstedt, "On achieving optimized capacity utilization in application overlay networks with multiple competing sessions," in *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, 2004.
- [6] M. Grottschel, L. Lovasz, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimizations*, Springer-Verlag, 1993.
- [7] G. Baier, E. Kohler, and M. Skutella, "On the k-splittable flow problem," in *Proc. of European Symposium on Algorithms*, 2002.
- [8] K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," in *ACM SIGCOMM 2000*, 2000.
- [9] P. Ramanathan C. Dovrolis and D. Moore, "What do packet dispersion techniques measure?," in *Proc. of IEEE INFOCOM*, 2001.
- [10] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and 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.
- [11] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts, "Competitive routing of virtual circuits with unknown duration," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1995.
- [12] B. Awerbuch, Y. Azar, and S. Plotkin, "Throughput-competitive on-line routing," in *IEEE Conference on Foundation of Computer Science (FOCS)*, 1993.
- [13] A. Goel, M. Henzinger, and S. Plotkin, "Online throughput-competitive algorithm for multicast routing and admission control," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1998.
- [14] M. Kwon and S. Fahmy, "Topology-aware overlay networks for group communication," in *ACM NOSSDAV*.
- [15] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *IEEE INFOCOM*, 2002.
- [16] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R. Y. Wang, "Overlay mesh construction using interleaved spanning trees," in *IEEE INFOCOM*, 2004.