Enabling Contribution Awareness in an Overlay Broadcasting System

Yu-Wei Sung*, Michael Bishop, and Sanjay Rao
ECE Department, Purdue University, 465 Northwestern Ave., West Lafayette, IN 47907 USA
sungy@purdue.edu, bishopma@purdue.edu, sanjay@purdue.edu
Phone: (765)494-3502, Fax: (765)494-0676

EDICS Category: 5-STRM

Abstract

We consider the design of bandwidth-demanding broadcasting applications using overlays in environments characterized by hosts with limited and asymmetric bandwidth, and significant heterogeneity in upload bandwidth. Such environments are critical to consider to extend the applicability of overlay multicast to mainstream Internet environments where insufficient bandwidth exists to support all hosts, but have not received adequate attention from the research community. We leverage the multi-tree framework and design heuristics to enable it to consider host contribution and operate in bandwidth-scarce environments. Our extensions seek to simultaneously achieve good utilization of system resources, performance to hosts commensurate to their contributions, and consistent performance. We have implemented the system and conducted an Internet evaluation on PlanetLab using real traces from previous operational deployments of an overlay broadcasting system. Our results indicate for these traces, our heuristics can improve the performance of high contributors by 10-240% and facilitate equitable bandwidth distribution among hosts with similar contributions.

I. Introduction

In the last few years, a peer-to-peer approach has emerged as a key alternative to enabling video broadcasting applications on the Internet [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. The key aspect that makes such an approach attractive is that a participant that tunes into a broadcast is not only downloading a video stream, but also uploading it to other participants watching the stream. Consequently, such an approach has the potential to enable ubiquitous Internet broadcasting, as greater demand also generates more bandwidth resources.

While the self-sustaining nature of peer-to-peer systems makes them attractive, it also presents a fundamental challenge. In particular, the feasibility of the architecture requires that the total upload bandwidth available from peers exceeds the download bandwidth demand. In mainstream Internet environments today, this is challenging given the majority of hosts are behind asymmetric connections (download $\gg$ upload) such as DSL and cable modem. In addition, the upload bandwidth of peers are highly heterogeneous due to different link technologies and different willingness to contribute. These factors have proved a first-order challenge to peer-to-peer broadcasting as indicated in a recent deployment experience [15], yet have received suprisingly little attention in the community.

The mismatch between upload and download bandwidth also impacts other peer-to-peer applications such as file download applications [18], [19]. However, the key aspect that distinguishes live broadcast applications from file download applications, and makes the problem much more challenging, is the stringent timeliness and real-time requirements. For example, file download applications can cope with bandwidth resource mismatch through longer download times. Further, they can benefit by the presence of ”seed” nodes that contain the entire file, and upload but do not download data. These approaches are simply not feasible in live broadcast applications, necessitating fundamentally different design approaches.

In this paper, we seek to enable broadcast applications in bandwidth-scarce, heterogeneous operating environments by addressing design challenges unique to live broadcasting. We consider a contribution-aware framework where nodes receive different levels of bandwidth based on the overall upload bandwidth available in the system, and the contribution of the node. Heterogeneity in node contribution may arise due to both inherent variation in node upload capabilities (Ethernet vs. DSL), as well as their different willingness to contribute bandwidth resources. To achieve these goals, we leverage the multi-tree framework [7] to enable application-level adaptation. While the multi-tree framework was originally proposed to improve resiliency, we use it to enable application-level adaptation and differential treatment.

A key element of the system is the distribution policy used to determine the bandwidth a node is eligible to receive, given its own contribution, and the overall system resource level. A simplistic solution is ”bit-for-bit” policies, where a node simply receives as much bandwidth as it contributes. However, this proves insufficient in today’s Internet environments, given there may exist hosts behind Ethernet capable of contributing much more than the source rate, while hosts behind asymmetric connections such as DSL and cable modem are precluded from receiving more than their upload capacities. We instead consider more generic policies, that ”tax” bandwidth-rich hosts to offer better performance to bandwidth-poor hosts [20]. To support these policies in a distributed manner, we design distributed heuristics for monitoring of overall system resources, differential and equitable distribution of bandwidth resources, and

application-level adaptation to changes in host contribution. While our system framework is motivated by [7], [20], to our knowledge this is one of the first and most comprehensive reports on design and implementation experience of overlay broadcasting systems, explicitly targeted at heterogeneous and bandwidth-scarce Internet environments.

We have conducted an evaluation of our contribution-aware broadcasting system on PlanetLab using traces from real overlay broadcast deployments. Our results show that our heuristics offer differential and equitable resource distribution when compared to a contribution-agnostic system. In particular, the 10th-percentile performance of high contributors (nodes contributing more than 175% of the source rate) is increased by 10-240% and variation of bandwidth received among nodes with similar contributions is reduced across our set of traces. Achieving these improvements does incur a $20 - 38\%$ decrease in the time between quality changes seen by a host and a minor increase in overhead, but achieves a 10-fold reduction in average time to recover from these changes for high contributors.

Section II describes the problem and challenges. Sections III and IV present the design of contribution-aware heuristics. Section V describes the process of implementing the multi-tree framework into the broadcasting system our implementation is based on. Section VI discusses our evaluation methods and metrics. Evaluation results are presented in Section VII. Section VIII discusses open issues and related work, and we conclude in Section IX.

II. Design Rationale

We begin by presenting a more formal definition of the problem being tackled (Section II-A). We then motivate the key components that our system requires, given the unique nature of broadcasting applications (Section II-B).

A. Broadcast User Model

A peer i in the broadcast receives and simultaneously forwards data. We assume that all peers are capable of receiving the full *source rate* S when the system is capable of providing it to them. This is reasonable, given many "broadband" users today have asymmetric connections with a reasonably large downloading capacity. Most DSL hosts would easily be able to receive S but not forward one full-rate video stream. In academic or business environments, symmetric connections (eg: Ethernet) are more common. Such hosts often could receive and forward many times more than S.

Each peer i may contribute a maximum upload bandwidth of F_i, the *forwarding bound*. F_i is the minimum of: (i) the willingness bound, which is the maximum upload bandwidth that peer (user) i is

TABLE I

COMPARISON OF DESIGN DECISIONS BETWEEN FILE DOWNLOAD AND LIVE VIDEO BROADCAST APPLICATIONS.

	File Download	Live Video Broadcast
Application Requirements	Non-interactive, timeliness not critical.	Bandwidth demanding, stringent real-time requirements.
Handling bandwidth-scarce regimes	Could cope through longer download time. Could leverage seed nodes which already have entire file and only upload.	Cannot compromise on delay requirements. Need a framework for application degradation. Synchronous/simultaneous nature implies no seed nodes that have entire content.
Incentive policies	Tit-for-tat schemes possible, users reward each other purely based on mutual benefit.	Timeliness requirement poses difficulties to always arrange A and B to both upload content to each other simultaneously. Consider overall contribution made to system while rewarding node.
Preventing saturation of outgoing access link	Saturation of parent's outgoing access link may lead to slower transfer time but acceptable.	Saturation of parent's outgoing access link impacts video quality all descendants see. May force users to quit.

willing to contribute; and (ii) the transient upload bandwidth capacity of i. The willingness bound is configured at join time whereas the transient capacity bound changes based on instantaneous network conditions and is unknown to the user. Consequently, F_i must be dynamically estimated, as described in Section II-B. We assume that F_i is non-zero – every peer will contribute *some* bandwidth upon request.

While F_i is the maximum bandwidth that a peer i may contribute, f_i ($\leq F_i$) is the *actual contribution* of i at a given instant. f_i may vary over the course of i's stay in the system due to changes in the number of peers i forwards data to. Our system determines the bandwidth r_i each peer is entitled to receive, based on its actual contribution f_i. Given that bandwidth received by a peer must be forwarded by other peers in the system, we have $\sum_i r_i = \sum_i f_i$. Further, in bandwidth-scarce environments, insufficient upload bandwidth exists to supply every peer with the full source rate, so $\sum_i f_i < N * S$, where N is the number of nodes. Lastly, we assume hosts honestly report their f_i and the bandwidth received. However, we believe our system can be easily integrated with recent research in distributed auditing and rating of nodes [21], [22], [23] to verify the claimed contribution of nodes.

B. Key Design Elements Motivated By Unique Nature of Broadcast Applications

Video broadcast applications impose stringent real-time performance requirements, involving timely and continuous delivery of streams of hundreds of kilobits per-second. These stringent real-time requirements distinguish them from file download applications such as BitTorrent [18] and Emule [19], requiring us to adopt fundamentally different design approaches to tackle the challenges of bandwidth-scarce and heterogeneous Internet environments (Table I):

• *Frameworks for application-level adaptation:* File download applications do not have a fixed "demand" for bandwidth, and could cope with the bandwidth-scarce regimes through slower download rate. Further,

given that file downloads are not synchronous, it is possible to leverage ”seed” nodes that contain the entire file, and only upload but do not download. In contrast, in video broadcast applications, each recipient needs a certain consistent download rate to obtain good performance, and the ”live” nature precludes the use of seed nodes. This motivates us to consider frameworks where nodes can gracefully adapt to the bandwidth availability in the environment through receiving video of degraded quality.

• *Global Vs. Pairwise Incentive Decisions:* File download applications like BitTorrent adopt a *tit-for-tat* strategy to incentivize peer contributions. In particular, a peer A which uploads a file segment to a peer B also requires B to simultaneously upload another segment to A. The strategy works well in peer-to-peer file download because there are no real-time requirements, and the order in which segments are downloaded is not important. However, this approach does not trivially extend to video broadcast because of the timeliness requirements involved, which means all hosts are likely to require the same segments at the same time. Thus, this leads us to explore an approach where the amount of bandwidth a node uploads to the system is used to decide the bandwidth it is eligible to receive, rather than local pair-wise decisions.

• *Preventing saturation of outgoing access link:* Typically, applications today rely on users to configure their upload bandwidth limits (i.e. willingness bounds). While incentive mechanisms encourage peers to allocate as much of their upload bandwidth as possible, the converse problem occurs when a peer advertises a willingness bound higher than its transient capacity. This may occur due to a user who over-estimates his connection capability, or due to transient network conditions. For file download applications, this problem is not critical, with nodes downstream of a misconfigured peer experiencing longer transfer time. In contrast, such degraded performance is not tolerable in live video broadcast. Figure 1 shows a DSL node with a upload capacity of 120 Kbps declares a limit of 450 Kbps. With a stream rate of 100 Kbps, it is over-saturated by supporting 4 nodes. Each downstream node receives a stream quality much below the original stream, prompting the user to leave the broadcast. To make matters worse, congestion on the DSL node’s outgoing access link may ultimately impact its own stream quality.

III. Design Overview

In this section, we provide an overview of key components leveraged to address the unique application requirement of video broadcasting applications described in Sections II-B.

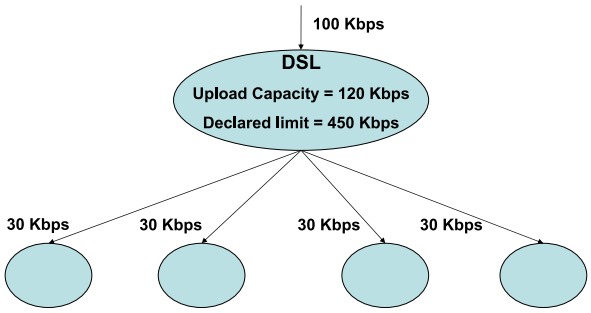


Fig. 1. A misconfigured DSL node.

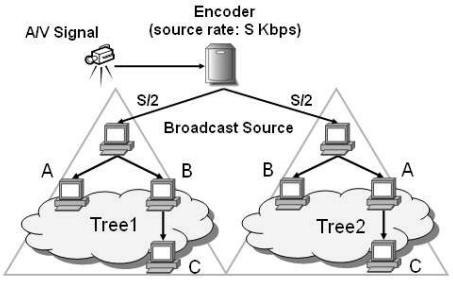


Fig. 2. A multi-tree broadcast with two trees.

A. *Multi-tree-based Data Dissemination*

In bandwidth-scarce environments, bandwidth available may be insufficient for all peers to receive the full source rate so we require a means by which hosts may receive and contribute graduated levels of bandwidth and transition smoothly as available bandwidth change. To realize these goals, we leverage the multi-tree data delivery framework [7], [11]. In this framework, participants self-organize into a forest of T trees rooted at a source. The source encodes video with a rate of S evenly into T stripes of size S/T, each distributed along one distinct tree. The quality a host gets depends on the number of stripes it receives. Typically, a layered codec based on Multiple Description Coding (MDC) [24], [25] is used to realize this goal. The trees are *interior-disjoint*; that is, a host i allocates F_i to only one tree but attempts to connect to all of the T trees. When F_i is normalized by S, we call the resulting value the *degree* of host i. For example, if $F_i = 300Kbps$ and $S = 400Kbps$, i's degree is $300/400 = 0.75$. We also define the *tree-degree* d_i to be $degree * T$, which is the maximum number of children a host can support in the tree it contributes. Figure 2 illustrates how the broadcast content is delivered with $T = 2$. Host A and B both have a degree of 1 and allocate their bandwidth in Tree2 and Tree1, respectively, where each can support two children (i.e. $d_i = 2$). C receives $S/2$ each from A and B to reconstruct the original content. While this multi-tree framework was originally proposed to improve resiliency [7], [11], we use it as a convenient building block to address issues regarding heterogeneity and bandwidth distribution.

This framework meets our needs, because it allows nodes to connect to a subset of trees and contribute in smaller bandwidth increments (i.e. stripes). Any node with a degree greater or equal to $1/T$ (i.e. tree-degree ≥ 1) is able to contribute. By setting T properly, we allow nodes with limited upload bandwidth to contribute, thereby spreading the forwarding load across all peers. Table II summarizes terminology introduced so far.

TABLE II

SUMMARY OF TERMINOLOGY.

S	Source stream rate
T	Number of overlay trees
N	Number of broadcast participants
F_i	forwarding bound, max bandwidth that can be forwarded = min(transient capacity, user configured willingness bound)
f_i	instantaneous bandwidth forwarded
r_i	bandwidth i is entitled to receive
d_i	tree-degree = max # of children that can be supported across all trees, computed as $F_i/(S/T)$

B. Bandwidth Distribution Policies

A key design consideration is the selection of policies for distributing bandwidth in the broadcast among participating hosts based on their contributions. Assuming there are N hosts, and host i forwards bandwidth f_i, our heuristics determine the bandwidth r_i each host is entitled to receive. Based on r_i, the multi-tree framework enables hosts to obtain different video qualities, by connecting to a subset of trees at a high priority. However, the granularity of distribution is limited by the number of trees, T, in the forest. A larger T enables greater granularity but may increase the control and coding overhead.

Our heuristics do not prescribe any particular bandwidth distribution policy; however, it is designed with the goal of providing a framework that can enable the implementation of a range of policies. One simple bit-for-bit policy is to require each node to forward as much as it receives, that is, $r_i = f_i$. Under this policy, it is straightforward for each node to determine the bandwidth it should receive, as the decision is easy to compute locally. However, this policy is restrictive in two ways. First, it does not account for the fact that nodes may contribute less bandwidth than the source rate. Further, it does not provide any incentive to a node to donate more than the source rate even if it is capable of doing so. This is an issue in Internet environments today. Consider the fact that Internet broadcasts typically involve a source rate of 300-400 Kbps, with a majority of hosts behind DSL and Ethernet. Hosts behind DSL can receive the source rate, but are not capable of forwarding it. Hosts behind Ethernet are capable of contributing much more than the source rate, and a policy such as $r_i = f_i$ neither utilizes the bandwidth, nor incentivizes them to contribute more. On the other hand, arbitrarily sophisticated policies may be extremely difficult to implement in a distributed fashion.

Instead, we consider a generic cost function of the form proposed in [20]:

$$r_i = \frac{1}{t} * f_i + \frac{t-1}{t} * \sum_i \frac{f_i}{N} \tag{1}$$

r_i is the bandwidth peer i is entitled to receive, f_i is the bandwidth i contributes to the system. N is

the number of participating peers. t is the "tax rate", which specifies a peer must contribute $t * r_i$ units of bandwidth to receive r_i unit of entitled bandwidth. t must be greater than 1. If $t = 1$, we have a simple bit-for-bit policy. If $t \leq 1$, no surplus exists and some peers will not receive their entitled bandwidth. r_i is the sum of two terms. The first term represents the minimum bandwidth a peer is entitled to receive by contributing f_i, and the second term is the average leftover bandwidth per node. By using a tax rate greater than 1, we are assured extra bandwidth in the system. For example, if t is 2, a peer i which contributes $2S$ will consume S from the system. The leftover resource S contributed by i is excessive. We aim to distribute all such excessive resources evenly among all peers, and this is represented by the second term. Since every byte of bandwidth received by a peer must be contributed by another peer(s). We can easily confirm that bandwidth is conserved by summing up both side of Equation (1) over all nodes, leading to $\sum_i r_i = \sum_i f_i$. For our later evaluation, we pick a tax rate of 2.

We will focus on (1), as it lends itself to implementation in a distributed fashion. Section IV-A.1 describes a distributed way to obtain system-wide estimates such as $\sum_i f_i$ and N.

C. Detecting and Preventing Saturation of Outgoing Access Links

As discussed in Section II-B, overestimating the upload bandwidth of a node could significantly impact the performance of its downstream users. The upload bandwidth that a node is capable of contributing at any time depends not only on its capacity, but also on background traffic or transient network congestion. Thus, we require mechanisms to dynamically estimate the maximum upload bandwidth that a node is capable of contributing, which in turn helps determine F_i. One potential approach involves leveraging ongoing work [26], [27] to estimate the available bandwidth between a sender and receiver. Such tools are designed for general scenarios, where the bottleneck link may be anywhere inside the network. We instead focus on techniques enabled by our specific context. In particular, we leverage the fact that when multiple downstream receivers simultaneously observe congestion, it is likely an indication that the parent's outgoing access link is saturated. We develop a technique where a node dynamically searches for its optimal forwarding bound, F_i, by starting with a low value and gradually increasing it. If the performance of all its children degrades, the node will assume it has overextended its ability to contribute and congested its outgoing access link. Upon such events, the node will reject one or more of its children and reduce F_i. Since congestion may be transient, the node should periodically attempt to raise F_i. To avoid destabilizing the system, such increase must be subject to a backoff – if an increase in F_i impacts performance of children, the node must wait for a longer period of time before making a subsequent attempt. In summary, each eligible parent adapts its F_i in the following manner: (i) increase

F_i to allow children to use leftover upload capacity, and (ii) upon congestion on the access link, drop a child and decrease F_i. We refer to the above technique as the **saturation detector**.

IV. System Design

In putting together the design, we had several criteria: (i) Contribution-Awareness, in that nodes must receive performance commensurate to their contribution; (ii) Good utilization, in that each node should contribute to the extent of its ability and willingness, with ideally no untapped bandwidth; and (iii) Stability in performance of nodes, without too much fluctuation in the bandwidth received.

To differentially treat a node based on its contribution in a forest, we consider the following problem: given a peer i, we wish to obtain a direct mapping between the actual amount of bandwidth, f_i, i contributes and the amount of bandwidth, r_i, the system should offer in return. Recall that i allocates its entire bandwidth F_i to only one tree, called i's *Contributor Tree*, but attempts to receive from **all** of the T trees. Extra bandwidth, if any, should be distributed evenly among participants once all of them get their deserved bandwidth. Equation (1) helps us obtain such mapping in a distributed fashion. We refer to r_i as the *Entitled Bandwidth* of i. The natural solution is to have i simply receive r_i by subscribing to $\lfloor \frac{r_i}{S/T} \rfloor$ trees as an *Entitled Node*. These trees are the *Entitled Trees* of i. However, there are two reasons why this may not suffice. First, each node can only be entitled to an integral number of trees. If r_i is not an integral multiple of the stripe rate, S/T, the fractional portion of r_i becomes superfluous. Second, there may be nodes whose r_i is larger than S, and they will not consume all of r_i entitled to them. Consequently, not all bandwidth is used by nodes entitled to it, and there exists some additional bandwidth in the system remained to be utilized. When a node's *Entitled* bandwidth is lower than the source rate S, it may utilize some of these additional bandwidth available in trees they are not entitled to. We refer to the additional bandwidth that nodes are not entitled to but utilize to reach the source rate as *Excess Bandwidth*, and nodes looking for or utilizing this bandwidth as *Excess Nodes*.

In summary, a broadcast participant may assume two "main" classes in the forest: it may be an *Entitled* node in some trees and an *Excess* node in some other trees. To treat different types of nodes with a better granularity, our system further classifies them and assigns them different priorities. When distributing system resources, our goal is to favor *Entitled* nodes over *Excess* nodes and evenly distributes the *Excess* bandwidth among all participants until they receive the source rate or no more resources remain.

Before presenting our design details, we want to make an important distinction between two concepts used in our multi-tree design: *join/subscribe* and *receive/connect*. A node *joins/subscribes* to a tree if it is aware of its participation within the tree, whether connected or disconnected, and a node *connects*

to/receives in a tree if it has attached to a parent and is receiving the data disseminated in the tree. We also define a *slot* as an allocated bandwidth of size S/T by a parent. A slot can be in one of three states: (i) occupied by an *Entitled* node, (ii) occupied by an *Excess* node, or (iii) unused.

A. Determining Number of Entitled Trees

To enable a node i to compute its *Entitled* bandwidth r_i using Equation (1), our system includes distributed mechanisms to periodically approximate the total resources utilized (i.e. $\sum_i f_i$) and the number of peers N. However, these global parameters may change any time due to group and network dynamics, leading to fluctuation of r_i. This may reduce the stability of the system because hosts overreact to system states. Therefore, our system includes a way to smooth out the impacts sudden changes in r_i have on the number of *Entitled* trees.

1) Distributed System Sampling: Collection of various system-wide parameters, for example, $\sum_i f_i$ and N, is necessary to compute r_i using Equation (1). We accomplish this by having each node in a tree periodically obtain the state of the subtree rooted at it and passing such information up the tree to the source. The source collects the state from each tree, generates system-wide information by aggregation, and propagates it down each tree to keep participating nodes informed about the system states in order to make cooperative decisions. To minimize message overhead while attempting to maintain a reasonable estimate of the transient system states, we choose a sampling period of 10 seconds.

Every 10 seconds, a node i informs its parent in each tree: (i) the bandwidth it is currently receiving from the tree, (ii) the total bandwidth received by its descendants, and (iii) the number of its descendants in each node class. The parent assembles these information from its children, aggregates with its own performance, and continues the process of passing information further up the tree. The source gathers the most recent updates from its children in each tree every 10 seconds, processes them, and sends down each tree a *control update* containing a monotonically increasing sequence number and the following system states: (i) the total contribution of the forest, $\sum_i f_i$, by summing up the bandwidth received by all nodes in the forest (ii) the total number of participants, N, measured by the total number of *Contributors* in the system (since each peer contributes in exactly one tree), and (iii) the number of *Excess* nodes connected to each tree. Since a host may receive control updates at different times from different trees it connects to, it extracts data from the update with the greatest sequence number.

2) Computing Number of Entitled Trees: After joining the system, a host i periodically (every 3 seconds) computes the number of trees it should be entitled to using the three-step process below:

a. Determine $T_{i_{sample}}$: A host i computes r_i based on Equation (1), using the most recent sample of

system states. To convert r_i to a raw computation of *Entitled* trees $T_{i_{sample}}$ for peer i, we normalize it by the size of a single stripe:

$$T_{i_{sample}} = \frac{r_i}{S/T}$$

The computation occurs more frequent (once every 3 seconds) than sampling (once every 10 seconds) because f_i is a transient value. Keeping the computation frequent enough enables a node to quickly adapt to the dynamics of its children and the system as a whole.

b. Smoothing $T_{i_{sample}}$: Since r_i can change abruptly at any time with f_i and N, it is advisable to smooth $T_{i_{sample}}$ to prevent the host from overreacting to peer and network dynamics. Two transitions could occur: $T_{i_{sample}}$ may either increase or decrease. It will increase either if more resources are utilized per node in the system, or if the node's contribution has increased. In either case, the change is likely to be relatively long-lived and should be quickly responded to. In contrast, the value will decrease with a drop in system resources or with departures of i's children. Children departures may be considered transient, as another child will be acquired quickly in bandwidth-scarce environments. Thus, we have implemented a smoothing scheme which tracks immediate increases in $T_{i_{sample}}$, but only gradually responds to decreases. To achieve this, i calculates its estimated number of *Entitled* trees, $T_{i_{est}}$, in this way:

If $T_{i_{sample}} < T_{i_{sample-old}}$,

$$T_{i_{est}} = (1-\alpha) * T_{i_{est-old}} + \alpha * T_{i_{sample}} \tag{2}$$

Else, $T_{i_{est}} = T_{i_{sample}}$

When the current sample, $T_{i_{sample}}$, is less than its previous sample, $T_{i_{sample-old}}$, we smooth using Equation (2) where $T_{i_{est}}$ is a weighted sum of $T_{i_{est-old}}$, the previous value of $T_{i_{est}}$, and $T_{i_{sample}}$. To put more weights on recent samples than on old samples, we set α to be 0.125, which from our experience has worked well. In [28], we have evaluated and shown the benefit of this smoothing heuristic.

c. Calculate $T_{i_{eff}}$: To further ensure the number of trees entitled to a node depends on the node's immediate history, $T_{i_{est}}$ is fed through a hysteresis processor, with a threshold of ± 0.1 around an integral tree value. The greater the threshold is, the more damping is imposed on $T_{i_{est}}$. The output of this processor is the effective number of *Entitled* trees, $T_{i_{eff}}$. For example, if the last $T_{i_{est}}$ calculated was 2.8, the current $T_{i_{est}}$ must exceed 3.1 to have a $T_{i_{eff}}$ of 3. Finally, we restrict $T_{i_{eff}}$ to the range $[1, T]$ and the resulting value is the number of trees to which i is entitled. It is lower-bounded by 1 since a host is always entitled to its *Contributor Tree* and upper-bounded by T because when $T_{i_{eff}}$ is greater than the total number of trees, i will simply be entitled to all trees.

B. Locating Excess Bandwidth

Since having a tax rate greater than 1 enforces each node to contribute more than its *Entitled* bandwidth, there will be leftover bandwidth in the system after nodes get their *Entitled* bandwidth. However, given the system does not know the maximum bandwidth F_i a node i will forward, it is difficult to determine the amount of these leftover resources and where they are located *until* they are found and utilized. Thus, we choose to have a host i periodically explore for free slots in trees where it is not entitled, as an *Excess* node, until successfully connected. We call these trees *Excess Trees* of i. Any successful connection represents a slot which is not currently used to satisfy demands from *Entitled* nodes and becomes a part of system's *Excess* bandwidth.

Having nodes actively probe for *Excess* bandwidth has an additional benefit. When a node joins the system, its contribution level is not known. A node cannot contribute without any demand for resources, but in a steady state this demand would not exist *until* it begins to contribute. In order to accelerate this bootstrap process, there must be an ongoing demand for bandwidth to enable under-utilized nodes to raise their actual contribution. However, such aggressive probing by *Excess* nodes may not be fruitful under bandwidth-scarce environments, as many of them may often compete with other nodes, including *Entitled* nodes, for the same slot, which may destabilize the tree structure. Thus, our system proposes a backoff scheme, in which an *Excess* node adaptively adjusts the aggressiveness of probing based on feedbacks received from the tree.

Backoff in Excess Trees: When an *Excess* node actively explores for *Excess* bandwidth, the attempt may fail due to (i) an inherent lack of resources (no free slots nor preemptable children, will be clear later) in the tree, or (ii) resources exist but the node is not able to locate them. In either case, the node presumes that the tree is saturated and will enter a phase of exponential backoff in which it waits for $t_{backoff}$ seconds before retry. Consecutive failures will result in an exponential increase in the backoff timer, which is computed as follows:

$$t_{backoff} = t_{base} * rand(\beta^k + T_{i_{excess}}) \tag{3}$$

where t_{base} is the backoff base, β is the backoff factor, k is the number of consecutive failures, and $T_{i_{excess}}$ is node i's overall number of connected *Excess* trees. ***rand(x)*** returns a random number in $(0, x]$. Currently, t_{base} and β are set to 5 and 2, respectively. Since our results show that the average reconnection time for low-contributing nodes is around 1 minute, these parameter values allow an *Excess* node to successfully connect in 3-4 attempts.

This backup algorithm improves system stability since there is less contention for slots in a tree. A

node *attempts* to connect to its *Excess* trees at a low priority level, implying it may take longer to connect to the tree, and even if it does, chances are it will quickly be displaced by a higher priority node. In addition, the heuristic scales the backoff timer based on $T_{i_{excess}}$ to improve stability further because it biases *Excess* nodes connecting to fewer *Excess* trees, which have a higher priority than those connecting to more *Excess* trees. The prioritization policy will be explained in detail in the next section. Finally, to prevent nodes from repeatedly contending for the same slot(s) in the future, we use a rand function to inject some randomness in the backoff computation. In [28], we have evaluated the backoff heuristic and shown it significantly improves system stability.

C. Contribution-Aware Node Prioritization

In order to provide differential treatment to nodes forwarding at different levels, we introduce the notion of a class-based design. In this design, we further distinguish an *Entitled* node by whether it contributes or not. A node in a given tree belongs to one of three classes, in decreasing order of priority:
***Entitled Contributor* (*Contributor*)**: A node entitled to the tree and contributes bandwidth of at most F_i.
***Entitled Non-Contributor* (*Entitled-NC*)**: A node entitled to the tree but contributes no bandwidth.
Excess: An *Excess* node is not entitled to the tree and contributes no bandwidth. It actively explores for a slot in the tree and is able to connect only if free slots or slots of lower priorities are available.

A host subscribes to multiple trees but may assume a different class in each tree. Nevertheless, a peer always joins exactly one tree as a *Contributor*. This allows all hosts, regardless of its contribution level, to be entitled to at least one stripe upon entering the system, which in turn guarantees them with some minimum quality.

To assign priorities by class, we implement a class-based prioritization. In this scheme, when a disconnected node of higher class cannot find an empty slot, it will displace/preempt a node of a lower class. That is, when disconnected, high *Contributors* may displace low *Contributors*, *Contributors* may displace non-contributors, whether *Entitled-NC* or *Excess*, and *Entitled-NCs* may displace an *Excess* node. This scheme provides incentives, since those who contribute more will reach the full source rate faster, and achieves even distribution of *Excess* bandwidth among hosts. In order to offer more stability/protection to nodes with higher priorities, when a node can not find an empty slot, it will preempt, among nodes it knows, the one with the lowest priority. Details of the scheme can be found in [28].

D. Multi-Tree Join Management

Upon joining the multi-tree broadcast, a host i contacts the broadcast source and retrieves the following information about the system: (i) the number of trees T, (ii) the source rate S, (iii) the total number of participating hosts N, (iv) the total contribution in the system $\sum_i f_i$, and (v) the number of *Excess* nodes in each tree. i will select, with higher probability, the tree containing fewer *Excess* nodes as its *Contributor Tree*. Without any knowledge of i's willingness bound, balancing the non-entitled resources (i.e. *Excess* and unused slots) across each tree at join-time is difficult. Our join mechanism strives to keep each tree balanced in resources by encouraging new hosts to contribute in the tree with fewer *Excess* slots, which implies a shortage of resources in the tree. At this point, i does not know how many trees it is entitled to since it has not begun to contribute. An optimistic decision could provide the host with more opportunities initially than it deserves. Thus we permit the host to initially join the remaining $T-1$ trees as an *Excess* node.

Note that the effective number of *Entitled* trees $T_{i_{eff}}$ computed by a host i may change upon every computation period. In case of an increase, among i's *Excess* trees, it picks one with the most *Excess* nodes and upgrades its class to *Entitled-NC.* On the other hand, in case of an decrease, among trees in which i is an *Entitled-NC,* it picks one with the fewest *Excess* nodes and downgrades its class to *Excess*. This process repeats until i is entitled to $T_{i_{eff}}$ trees.

E. Detecting and Preventing Node Saturation

To dynamically adapt a *Contributor* i's degree to network conditions, i must infer whether its current forwarding bound F_i, and therefore the tree-degree d_i (recall $d_i = F_i/(S/T)$), accurately reflects the capability of its outgoing access link. This is achieved by i's **saturation detector**. If d_i is too low, the parent's bandwidth resource is under-utilized and d_i should be increased. However, there is no explicit signal to indicate this case. In fact, the transient amount of spare upload bandwidth is unknown until it is *used* to forward data to a child. To deal with this, the detector starts with a low d_i. If no congestion event occurs at the current level, the detector performs an *adoption-experiment* to see if d_i can be increased. The steps are as follows: before each experiment, a parent i launches an *adoption-timer* T_A. This is how long the system must wait between completion of one experiment and the preparation of a new experiment. The intuition is to begin with a small T_A, to ensure a child can quickly ramp up to its limit until d_i reaches the user defined limit. Therefore, T_A is set initially to 6 seconds. Upon expiry of T_A, i *prepares* an experiment by incrementing d_i by 1. After a while, the experiment starts when an "additional" child connects to i, at which point a *detection-timer* T_D is launched. T_D is the estimate of the time it takes

for the impact of a traffic change to be detected by the parent. The experiment is regarded successful if no congestion occurs upon expiry of T_D, at which point the parent schedules a new T_A and so on. When an experiment is successful, the detector optimistically assumes that i has plentiful capacity left, in which case adoption experiments should be conducted more frequently, by reducing T_A by $1/3$.

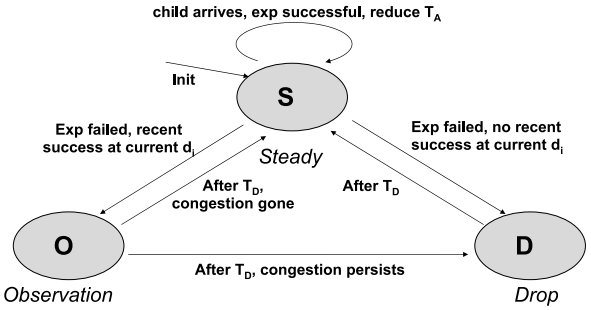
On the other hand, if d_i is too high, the parent's outgoing access link will be congested when adopting more children or when cross traffic increases. To detect a *congestion event*, the saturation detector running at the parent relies on periodic reports from its children. A child periodically(every 3 seconds) reports to its parent its received bandwidth in the tree, smoothed over recent bandwidth samples, and an instantaneous per-hop loss rate to indicate whether the overlay link between the parent and the child experiences a high packet loss. The parent i collects reports from all children and infers whether the overlay link to a child is congested. Specifically, if only one overlay link among several is identified congested, the congestion is likely to be near the child – the parent will not react to it. If all overlay links are congested, it is likely all children are behind one or more shared physical bottleneck links. In fact, a bottleneck is identified by the parent if the instantaneous per-hop loss rates of all children are above a certain threshold (20% in our implementation) and no child's bandwidth is above 80% of the parent's bandwidth. In this case, the parent conservatively assumes the bottleneck is on its access link and signals a congestion event since it is hard to distinguish between whether the bottleneck is near the parent or near each child. In addition, the result of a congestion event on the outgoing access link could also degrade the parent's downloading performance due to increasing protocol control traffic. To handle this, if the parent's performance is degraded significantly (40% in our design), it is also determined that a congestion event occurs.

Figure 3 illustrates the operation of the degree detector using a state machine. There are 3 states represented by ovals: steady-state(S), drop-state(D), and observation-state(O). Each arrow is labeled with a corresponding event that triggers the state transition. Upon join time, a *Contributor* i starts from the S state and a d_i of 1. i performs repeated adoption-experiments, and reduces T_A upon successful experiments as described previously. However, if at any time a congestion event occurs, the experiment is considered failed, and there are two possible states we could go to: D or O.

First, if no recent experiment succeeds at a level greater than the current d_i, the parent is likely over-extending its upload capability and we enter the D state, which involves the following actions: drop the lowest-priority child, decrement d_i, and double the adoption-timer T_A. We double T_A because once closer to the capacity of the parent, congestion events occur more frequently and it is important to prevent oscillation between an acceptable and unacceptable degree value. Doubling T_A prevents oscillation around an acceptable degree threshold, and allows the system to still probe for higher values after longer

periods of time. If successive adoption-experiments increase d_i over the parent's transient upload capacity, adoption-experiments will fail repeatedly, and T_A will become large. As a result, the parent will conduct adoption-experiments only occasionally and is likely to continue operating at the current degree level for most of its lifetime.

On the other hand, if the current d_i is smaller than the tree-degree of a recently successful experiment, it may be caused by increased transient background traffic. Hence, we transition to the O state. In the O state, we start a detection-timer and record the number of congestion events by taking active measurements of the child states. Once the detection-timer expires, we assume any transient increase in background traffic goes away and monitor the fraction of congestion events. If it is greater than some threshold (50% in our implementation), the parent enters the D state. Otherwise, it goes back to the S state without reacting to transient congestion periods.



State	Action
S	T_A sec since prev exp ends, prepare new exp, i.e. d_i++
D	Drop lowest-priority child, d_i−−, double T_A
O	Monitor congestion events for T_D sec

Fig. 3. State machine for the saturation detector.

Finally, in the D state, the parent starts the detection-timer and ignores further congestion until the timer expires. This prevents cascaded drop-actions from a single congestion event and gives the network sufficient time to recover from the congestion before further action is taken. At the end of the D state, we go back to the S state, schedule a new adoption-timer and simultaneously look for further congestion.

V. Implementation

To evaluate our heuristics on a real system, we have chosen to implement them on the ESM Broadcasting System [29]. ESM is a functional overlay broadcast application that has been used in academic conferences and workshops. The ESM client code is approximately 43,000 lines. The base ESM system uses a single-tree overlay to delivery broadcast content. This section summarizes ESM and describes how we extend it to use multiple overlay trees.

The ESM protocol [15] relies on a gossip-based group membership process to create an overlay broadcast tree among participating peers. Each node maintains knowledge about other members and continuously monitors its own performance in the tree. If a node's parent leaves the broadcast, or the performance received from its parent is unsatisfactory, it switches to some other parent. The reader is

referred to [15] for further details.

We employ a minimalist approach in extending the code base to the multi-tree framework. In our multi-tree implementation, we add a layer called the *Multi-Tree Agent* (MA) which contains an array of *Single-Tree Protocol Agents* (SPA, the single-tree protocol is ESM in our case). Each SPA is associated with one tree in the forest. The MA maintains global states, makes global decisions, multiplexes and de-multiplexes outgoing and incoming messages for a given tree to the associated SPA. Each tree operates independently and in parallel, interacting with the MA but not with the other SPAs. Finally, we incorporate our contribution-aware heuristics introduced in Section IV into the Multi-tree ESM Broadcasting System.

VI. Experimental Evaluation

We have evaluated our contribution-aware heuristics with a view to answering the following questions:

• How effective are they in ensuring good overall performance by utilizing the heterogeneous nature in the upload bandwidth of nodes in the system?

• How effective are they in offering differential and equitable performance to nodes based on their contributions?

• How stable is the resulting system, in terms of frequency of changes in the number of connected trees?

• How effective is the mechanism to detect and prevent node saturation?

To answer these questions, we have conducted experiments on PlanetLab employing real traces of join/leave dynamics to compare the following two systems:

Cont-Agnostic: This refers to multi-tree ESM without any contribution-aware heuristics. Further, the only possible preemption is that a *Contributor* can preempt an *Entitled-NC* or *Excess* node. This system is very similar to SplitStream [7] and CoopNet [11].

Cont-Aware: This refers to multi-tree ESM with the contribution-aware heuristics from Section IV.

A. Performance Metrics

We evaluate our system based on the following metrics:

• **Bandwidth**: For each node, we measure the mean application throughput in Kbps over its lifetime. To maximize quality of the received video, this metric should be as close to the source rate as possible.

• **Time Between Tree Reductions**: This metric measures the impact of our heuristics on the stability of the system using the average time between reductions in the number of connected trees a node experiences. The assumption is that the user perceived quality is dictated by the number of trees a node is connected to so a reduction degrades the user perceived streaming quality. Reductions should not be frequent so

this metric should be as large as possible, or application performance will be inconsistent. However, we should be careful while interpreting this metric as it does not distinguish different types of reductions. For example, in a forest of four trees, a reduction from four to three trees is treated the same as a reduction from one to zero trees. Thus a reduction does not necessarily mean a user sees bad quality.

• **Reconnection Time**: When a node is disconnected from a tree, it should reconnect quickly. This implies preempting nodes of lower priority or locating a unused slot. Time between tree reductions measures how frequently a user experiences a dip in performance; this metric describes how long such dip persists.

• **Control Overhead**: In additional to regular data traffic, control messages are sent regularly to construct and maintain the overlay structure, as well as support various heurstics. This metric measures the control traffic in Kbps.

In addition to these metrics, we also considered the **utilization** metric in [28].

B. Experimental Methodology

Our study of the two video broadcast systems is conducted based on twenty minute segments of real-world traces obtained from previous operational deployments of the ESM Broadcasting System [15]. Table III summarizes the characteristics of the trace segments. The table shows the percentage of hosts behind high-speed and low-speed connections (typically, Ethernet and asymmetric DSL/cable connections, respectively), and the *Resource Index* (RI) [15] for each trace. The RI is the ratio of the total forwarding capacity in the system to the bandwidth required for all hosts to receive the full source rate. We classify traces with an RI above one to be bandwidth-rich and bandwidth-scarce otherwise. The table also shows the group dynamics patterns using the average and median stay time of nodes in the traces, and the number of joins and leaves in the 20 minute segments.

The primary trace we use for evaluation is the *Slashdot* trace, which is from a bandwidth-scarce broadcast to an interest-group where the majority of hosts are behind DSL/cable, and has a high churn rate with median stay times less than 180 seconds. *SIGCOMM2002* and *SOSP2003* are broadcasts of conferences, and thus have a much larger fraction of hosts behind high bandwidth university machines; as such, they represent bandwidth-rich environments for comparison. *GrandChallenge* is a broadcast of a vehicular competition, and *Rally* refers to a broadcast of an election campaign. These traces represent bandwidth-scarce environments. We focus our evaluation on the *Slashdot* trace and use other traces to study how sensitive our systems are to various operating environments. We emulate the traces by mapping each client to a PlanetLab host and use the same client join/leave patterns as in the trace segments to drive the experiment. Furthermore, we emulate DSL/cable and Ethernet hosts with a degree

TABLE III

CONSTITUTION OF HOSTS IN A 20-MINUTE SEGMENT FOR EACH REAL-WORLD TRACE.

Broadcast	Avg RI	Low Spd(100Kbps)	High Spd(10Mbps)	Avg Stay	Median Stay	Tot Nodes	Peak Grp Size	Joins	Leaves
SIGCOMM2002	1.32	34%	66%	1065 secs	1200 secs	88	78	12	10
SOSP2003	1.31	47%	53%	672 secs	173 secs	95	51	54	38
Rally	0.96	73%	27%	682 secs	362 secs	401	239	214	154
Slashdot	0.87	66%	34%	593 secs	168 secs	328	156	185	160
GrandChallenge	0.51	88%	12%	724 secs	412 secs	281	149	143	103

of 0.25 and 2, respectively. Since PlanetLab hosts are behind high-speed Ethernet connections and access link congestion rarely occurs, we *disable* the saturation detector when evaluating the two broadcasting systems from Section VII-A to VII-E. In Section VII-G, we evalute the saturation detector, by setting up standalone experiments involving an actual DSL host and several PlanetLab hosts.

For each experiment, four multicast trees are formed. Although one could improve the granularity of bandwidth distribution by using more trees to produce smaller stripes, the network and video-codec overheads increase with the number of trees. We consider four trees small enough to be efficient, while large enough to provide reasonable flexibility in varying the bandwidth. We use a source data rate of 400 Kbps, a typical size of streaming videos on the Internet [15]. The source streams a stripe of 100 Kbps to each tree. The clients already present before the start of the segment join the broadcast in a burst and begin contributing in the their respective *Contributor* trees. We allow them 2 minutes to reach a steady state, after which the rest of the clients follow the join/leave patterns in the trace for the next 20 minutes, and experimental data is collected over that period.

We consider hosts with mean contributions greater than 700 Kbps to be *High Contributors* (HC) and those with mean contributions between 75 and 100 Kbps to be *Low Contributors* (LC). This splits hosts with various contribution levels into two groups and helps us evaluate them separately. Each result is aggregated or averaged over three runs with a consistent set of PlanetLab machines. When presenting results, we filter out hosts which stay for less than 2 minutes. We will study the impact of node stay time in Section VII-B.

VII. EXPERIMENTAL RESULTS

We begin by showing the behavior of a typical host under the *Cont-Aware* system in a bandwidth-scarce environment using the *Slashdot* trace in Section VII-A. Next, under the same setting, we compare the performance and average time between reductions in the number of trees of hosts in *Cont-Aware* to those in *Cont-Agnostic* in Section VII-B and Section VII-C. The overhead of contribution-aware heurstics

is studied in Section VII-D. Section VII-E explores how *Cont-Aware* behaves in different operating environments and tax rate t. Section VII-F studies how *Cont-Aware* copes with hosts behind network address translators (NATs) and firewalls. The saturation detector is disabled initially and is evaluated alone in Section VII-G. We have also evaluated other system components such as smoothing, backoff, and load balancing in [28].

A. Results with a Typical Run

Figure 4 shows the performance of a typical high contributor in our system. The node begins by making zero contribution and connecting to its *Contributor* tree. Over the next minute, the number of children the node supports goes from zero to eight. As the number of adopted children increases, the number of successfully connected trees also increases quickly, as the node becomes entitled to them. The actual performance fluctuates due to the fact that ESM uses non-blocking TCP to transfer data across each overlay link, leading to burstiness on the received bandwidth. Note that because we smooth away transient drops in contribution, the sudden loss of children between 255 and 300 seconds does not impact performance, and the node quickly acquires new children. The node is briefly disconnected from one tree at 250 seconds as shown by a 100 Kbps dip of the solid line. This is due to the departure of the node's parent. However, because the node is contributing significantly to the system, the recovery time is very brief – the node finds a new location in the tree in under 6 seconds.

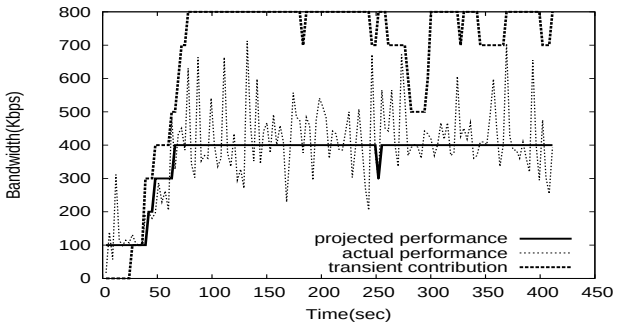


Fig. 4. Behavior of a typical high contributor under *Cont-Aware*. The top curve shows the bandwidth contributed, the solid curve shows the *Entitled* bandwidth, and the dashed line shows the actual bandwidth received.

B. System Performance and Utilization

In this section, we would like to evaluate how well *Cont-Aware* leverages the system resources and distributes them based on the contribution of each host as compared to *Cont-Agnostic*. In particular, hosts with similar contributions should see similar performance; hosts with higher contributions should see better performance than those with lower contributions.

Figure 5 plots the cumulative distribution of the mean session bandwidth of high contributors for the two schemes: *Cont-Aware* and *Cont-Agnostic*. There are two curves, each corresponding to one scheme. The y-axis is the CDF and the x-axis is the mean bandwidth ranging from 0 to 400 Kbps (i.e. source rate). The

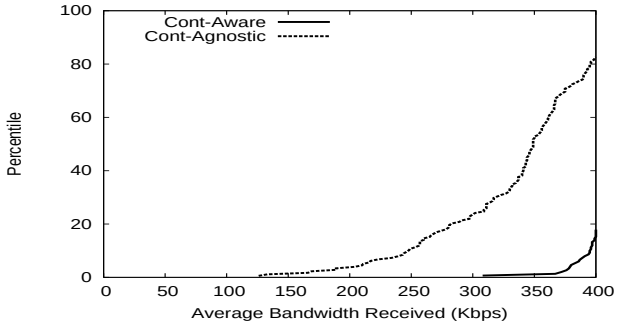


Fig. 5. Cumulative distribution of average received bandwidth for high contributors.

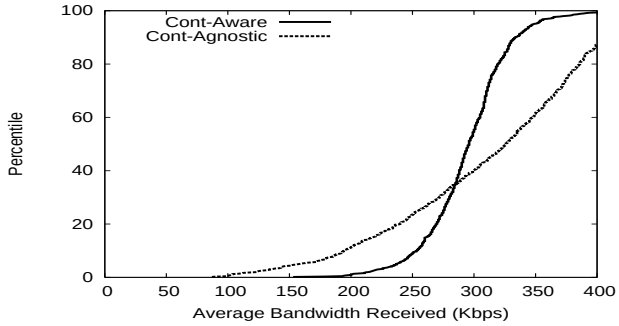


Fig. 6. Cumulative distribution of average received bandwidth for low contributors.

more a curve is toward the right, the better the overall performance is. *Cont-Aware* significantly improves the performance of high contributors with 80% of them receiving the source rate of 400 Kbps. *Cont-Agnostic* however allows only 20% of high contributors to receive the source rate. Furthermore, almost all high contributors under *Cont-Aware* obtain bandwidth greater than 350 Kbps whereas *Cont-Agnostic* does much worse, with only half of high contributors receiving more than 350 Kbps. By prioritizing high contributors, *Cont-Aware* allocates about two more stripes to each high contributor than *Cont-Agnostic*.

While Figure 5 plots the mean bandwidth CDF for high contributors alone, Figure 6 plots the same type of graph, but for low contributors. With *Cont-Agnostic*, almost all low contributors receive anywhere from 100 Kbps up to the source rate. *Cont-Aware* reduces this spread to 200-350 Kbps, bringing the performance of all low contributors toward the mean. This shows *Cont-Aware* enables nodes contributing similarly to receive similar bandwidth. To quantify this observation, we compute the mean and standard deviation of both curves and find that although low contributors in both schemes receive a mean bandwidth around 300 Kbps, with *Cont-Aware*, the standard deviation significantly drops from 80.5 to 34.8.

When looking at Figure 5 and 6 together, we see *Cont-Agnostic* gives high and low contributors a similar allocation pattern while *Cont-Aware* treat high contributors more favorably. Figure 5 also suggests *Cont-Aware* reduces the performance spread for high contributors. Furthermore, all low contributors under *Cont-Aware* receive at least one stripe of 100 Kbps. Thus we conclude that our contribution-aware heuristics achieve equitable and differential distribution of bandwidth based on nodes' contributions while offering some minimum guarantee on performance for low contributors. This offers incentives to nodes to contribute more and keep low contributors stay interested in the broadcast.

One question is whether it is possible to make the distribution among low contributors in *Cont-Aware* even more equitable, in which case most of them should receive closer to the average bandwidth of 300 Kbps – for example, 8% of the low contributors receive less than 250 Kbps. We see various reasons for

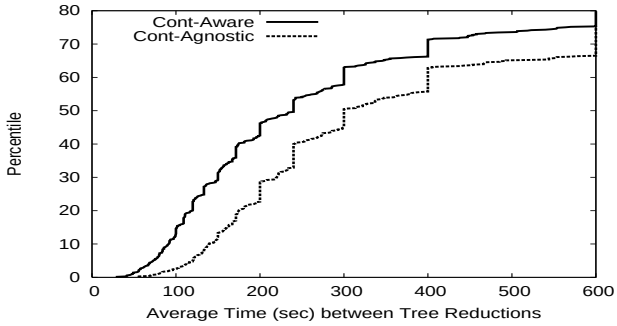


Fig. 7. Cumulative distribution of time between tree reductions for all nodes.

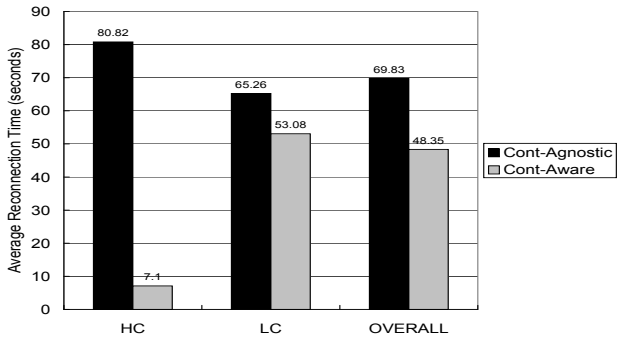


Fig. 8. Average post-preemption reconnection time in seconds for nodes in different contribution levels.

this. First, we are limited by the granularity imposed by the multi-tree framework, and more equitability could result with more trees. Second, some clients are limited by the bandwidth near them – further, they could be artifacts of our experiments as several clients may be mapped to the same PlanetLab machine and compete for incoming bandwidth. Third, there are convergence issues: short-lived low contributors do not remain in the system long enough to connect to their *Excess* trees, and due to the distributed nature of the system, resources are not always quickly located. In an extreme case, an *Excess* node which fails frequently on consecutive connection attempts may work up to a large backoff time, meaning they may make no further attempt to acquire a parent before they leave the system.

Finally, we have considered the *utilization* metric in [28]. In bandwidth-scarce *Slashdot* environment, *Cont-Agnostic* utilizes 95% of the resources in average, whereas the utilization of *Cont-Aware* is about 93%, demonstrating that our heuristics do not adversely impact the efficiency of ESM in locating and leveraging the available resources despite with numerous backoffs and preemptions taking place.

C. Time between Tree Reductions

We wish to show a node's received bandwidth is not frequently interrupted by measuring the time between reductions in the number of connected trees. Figure 7 shows the CDF of time between reductions in the number of connected trees for all nodes. We truncate the x-axis at 600 seconds, since nodes with fewer than one reduction in 10 minutes are considered stable. The higher the curve, the less stable the system is, since a greater percentage of nodes experience a smaller time between reductions in the number of connected tree. Although *Cont-Aware* appears to produce less consistent performance for all nodes, it does not necessarily imply users see bad performance due to two reasons.

First, the curve does not distinguish between different types of reductions. For example, a reduction from four to three trees is treated the same as a reduction from two to one trees. Table VII-C shows a breakdown of different types of reductions in the number of connected trees. We see that for *Cont-Aware*,

Reduction	*Cont-Agnostic*		*Cont-Aware*	
from→to	LC	HC	LC	HC
$1 \rightarrow 0$ trees	0.8%	0%	0%	0.1%
$2 \rightarrow 1$ trees	4.0%	3.1%	2.1%	0.3%
$3 \rightarrow 2$ trees	15.6%	13.8%	31.6%	2.5%
$4 \rightarrow 3$ trees	29.6%	33.1%	16.3%	47.1%

TABLE IV

BREAKDOWN OF DIFFERENT TYPES OF REDUCTIONS IN THE NUMBER OF CONNECTED TREES.

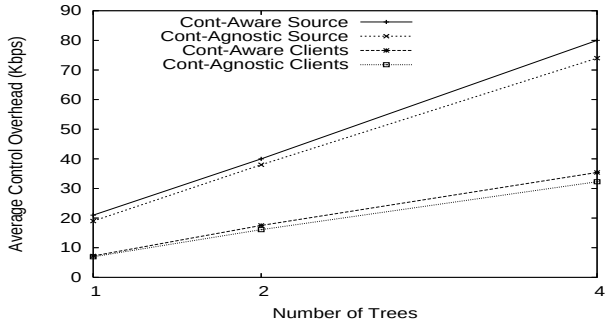


Fig. 9. Average control overhead for source and clients in both systems with T = 1, 2, and 4.

only 2.5% of reductions are from two to one trees and virtually none from one to zero trees whereas for *Cont-Agnostic*, almost 8% of reductions are of these types. We can also observe that almost all reductions high contributors in *Cont-Aware* experience are from four to three trees, which have very little impact on application performance. In contrast, such preferential treatment for high contributors is not obvious under *Cont-Agnostic*. Second, we find that 90% of reductions result from preemptions rather than from parent departures, and *Cont-Aware* allows preempted nodes to reconnect much faster.

By considering node contribution, our results show that the reconnection time for both high and low contributors are reduced, with a significant improvement for high contributors. In particular, the reconnection time of high contributors in *Cont-Aware* is only 1/11 of that in *Cont-Agnostic*. In *Cont-Agnostic*, a node which is preempted cannot preempt another node. In contrast, *Cont-Aware* establishes finer prioritization levels among nodes, and a preempted node can often quickly find a new location in the tree, reducing the cost of preemption. We also analyze the number of preemptions by preemption types and the average reconnection time following each type; the result can be found in [28].

D. Overhead

Figure 9 plots the average control overhead seen by the participants as a function of the number of trees T. There are four curves - the top two curves are the overheads seen by the source with *Cont-Aware* and *Cont-Agnostic*, while the lower two curves are the average overheads seen by clients with *Cont-Aware* and *Cont-Agnostic*. Overall, the additional overhead introduced by *Cont-Aware* is low. The overhead itself does increase linearly with T for both the *Cont-Aware* and *Cont-Agnostic* schemes. However, the overall overhead for clients is about 30 Kbps which is still much smaller than the typical data rates. Further, we believe several optimizations are feasible to reduce the overhead of the base ESM system itself, as well as the overhead due to introduction of multiple trees in ESM.

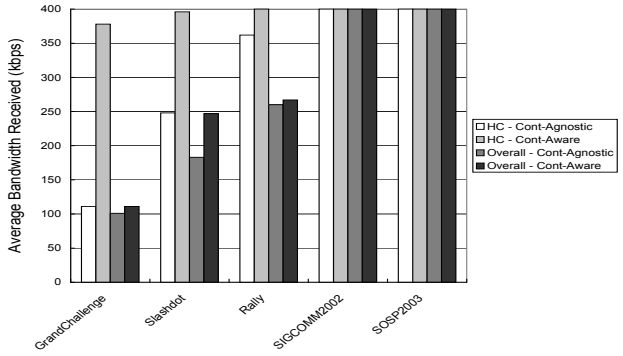


Fig. 10. 10th-percentile of received bandwidth for high contributors and all nodes in each trace.

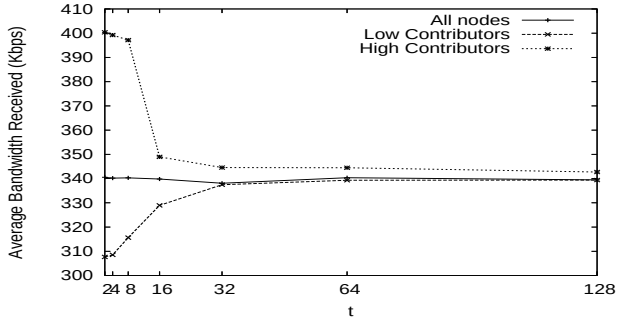


Fig. 11. Sensitivity of bandwidth received to tax rate(t).

E. Sensitivity Study

In this section, we evaluate our contribution-aware heuristics under environments with varying resource levels and churn rates by using five different traces. We include *Slashdot* among these for comparison to results in previous sections. Figure 10 shows the 10th-percentile performance of the entire set of nodes and high contributors across each trace for *Cont-Aware* and *Cont-Agnostic*. That is, 90% of all nodes see better performance than the numbers presented by bars. The traces are ordered based on their RIs, with the lowest RI at the very left. The three traces on the left are bandwidth-scarce whereas the two on the right are bandwidth-rich. Each trace has 4 bars, with 2 bars for high contributors and 2 bars for all nodes in *Cont-Aware* and *Cont-Agnostic*. For bandwidth-scarce traces, our heuristics offer improved tail performance for all nodes and high contributors alone. The significant improvement for high contributors confirm that they are prioritized for resources in *Cont-Aware* whereas improvement for all nodes implies their received bandwidth is pulled toward the mean. Among bandwidth-rich traces, we see similar performance – everyone successfully receives the source rate. We also study in [28] the sensitivity of time between tree reductions and reconnection time to these traces, which we omit here for brevity.

Sensitivity to tax-rate t: All our results thus far have assumed a tax rate $t = 2$. Figure 11 studies sensitivity to t for the *Slashdot* trace. There are 3 curves, each corresponding to the entire node set, high contributors and low contributors alone. Each curve plots the average bandwidth seen by nodes in that class against t. The average bandwidth seen by the entire node set remains unaltered with t. However, the difference in performance of the high and low contributors becomes more significant for smaller t.

F. Performance under NATs and Firewalls

A practical aspect that complicates contribution-aware heuristic design is that a large fraction of nodes today are behind NATs (Network Address Translators) and firewalls. In this section, we evaluate the

performance of *Cont-Aware* in such environments. We use the *Slashdot* trace as before, but assume 55% of hosts are behind NATs. We emulate the effect of NAT/firewall using a methodology as in [30]. The base ESM system ensures nodes behind NATs support public hosts as children, though nodes behind NATs cannot be supported [15].

Figure 12 studies the performance of *Cont-Aware* in the presence of NATs. The nodes are in four classes depending on whether they are behind NATs or not, and depending on whether their degree is 0.25 or 2, i.e. whether the maximum they will contribute is 100Kbps or 800Kbps. The graph has four sets of bars corresponding to these node classes, with each set containing two bars which show the average bandwidth forwarded and received for nodes in that class. Overall, it shows that the bandwidth resources contributed by nodes behind NATs (f_i) is only slightly lower than public nodes, and consequently the performance seen by the nodes is not significantly impacted. This result shows *Cont-Aware* works well in the presence of NATs. Further improvements may be possible based on the type of NAT (full-cone or symmetric), and allowing public hosts to prefer NATed nodes as parents during the selection process [15], [30]. We defer such study to future work.

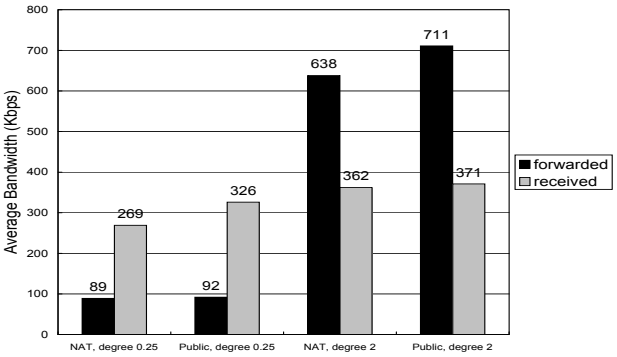


Fig. 12. Performance with NATs and firewalls. Degree 0.25 and 2 indicate that the maximum the nodes will contribute is 100Kbps and 800Kbps respectively.

G. Dynamic Detection of Node Saturation

In this section, we evaluate the benefit of incorporating the saturation detector into a parent. We want to show that when the saturation detector is enabled for a misconfigured parent, instead of adopting as many children as the user-defined willingness bound allows, the parent will choose to support only at a level where all children see decent performance. To show this, we set up a testbed consisting of a machine directly connected to the Internet via a DSL connection, and 9 PlanetLab hosts. The outgoing bandwidth of the DSL host is 384Kbps. We use one PlanetLab node as the source to deliver a source stream of 400Kbps to 4 trees (100Kbps stripe to each). The DSL host is configured with d_i = 8, while it can only support at most 3 children ($\lfloor 384/100 \rfloor = 3$). We set up a scenario where in the DSL host's *Contributor* tree, the other 8 PlanetLab hosts all try to connect to the DSL host as *Entitled-NCs* initially and leave the system 17 minutes later. We run two experiments under this scenario, one with the saturation detector disabled and the other with it enabled.

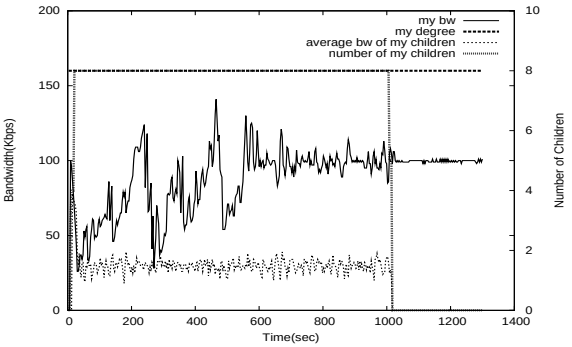


Fig. 13. A misconfigured DSL parent with saturation detector disabled.

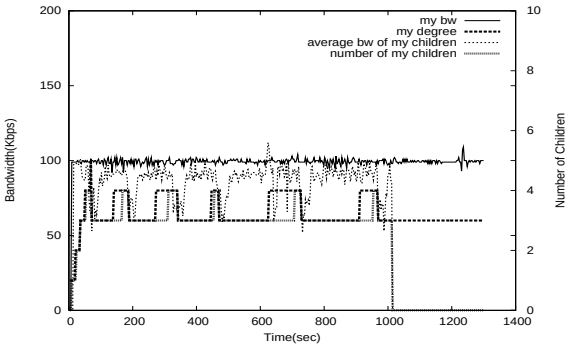


Fig. 14. A misconfigured DSL parent with saturation detector enabled.

Figure 14 shows the performance when the saturation saturation detector is disabled. The x-Axis is time since the start of the broadcast, the left y-axis is the bandwidth, and the right y-axis is the number of children. The figure shows (left y-axis) the bandwidth received by the DSL parent (solid line in the middle) and average bandwidth received by children (dotted line near the bottom) in the parent's *Contributor* tree over time. The thick and the dotted lines on the top (right y-axis) show the tree-degree d_i of the DSL host and the number of children it adopts over time, respectively. Since the saturation detector is disabled, d_i remains constant and all 8 children connect to the DSL parent. This results in the DSL node being over-saturated, which impacts its own performance, and the performance of its children. None of the children receive higher than 50Kbps, and the average bandwidth across children is less than 40Kbps. This is far below the expected 100Kbps stripe rate.

Figure 14 shows the performance when the saturation detector is enabled. The DSL node starts with d_i = 1 and gradually increases d_i, consequently acquiring more children. When the 4th child is adopted at the 75th second, a congestion event occurs as shown by the sudden dip in the average children performance from 100Kbps to 60Kbps. In response, d_i is decremented by 1 and a child is dropped. After a while, the parent again raises its degree to 4 at the 150th second and the same sequence of events follows. In fact, the DSL node supports only 3 children most of the time. d_i is raised to 4 only occasionally, and the interval between successive experiments increases, due to the increase in the adoption timer T_A. As a result, the performance of the children remains close to 100Kbps most of the time.

VIII. Discussion and Related Work

In this section, we discuss related work, and open issues.

Multi-tree Vs. Mesh: In this paper, we have focused on tree-based approaches for video delivery. This approach has been widely studied [15], [4], [6], [8], [2], [3], [10], [13], [11], [7] with nodes in the structure

having well-defined ”parent-child” relationships, and with each data packet being disseminated using the same structure. More recently, an alternate class of approaches based on meshes has emerged [31], [32], [17], [33]. Here, no explicit structure is maintained - instead each node maintains a set of partners, and periodically exchanges data availability information with the partners. A node may then retrieve unavailable data from one or more partners, or supply available data to partners. Both tree-based and mesh-based overlays have shown their success in practical deployments, and researchers are studying trade-offs between the approaches [34], as well as design of hybrid techniques [35]. We believe the notion of contribution-awareness however has not received sufficient attention in either class of approaches, and it would be interesting to extend the techniques in the paper to mesh-based designs.

Peer-to-Peer Vs. Hybrid Architectures: One approach to address issues with bandwidth-scarce regimes is to adopt hybrid architectures that seek to augment the bandwidth resources of application end-points participating in the application with infrastructure resources where available. In contrast, our focus is on ubiquitious broadcasting involving a purely application end-point architecture, which relies exclusively on bandwidth resources at application end-points. Our approach may also help hybrid architectures by allowing graceful degradation of performance when infrastructure resources are unavailable or insufficient.

Use of MDC: Our paper assumes the use of Multiple Description Codes (MDC) for video streaming. MDC technology is an active area of research [24], and still under active development. Further experience with real MDC implementations, and an understanding of the overheads they introduce is required before they may be used, and we defer this to future work. That said, the focus of the paper is on contribution-awareness, and the overhead associated with MDC applies both to the base *Cont-Agnostic* system (adapted from [7], [11]), and our proposed *Cont-Aware* heuristics.

Incentive Mechanisms: A wide body of work has looked at incentive mechanisms in the context of file-sharing applications (e.g. [36], [37], [21], [23], [22], [38]). These works do not directly apply to broadcasting applications given the real-time nature of the applications, and do not consider issues such as application-level adaptation, prioritization in tree placement, bandwidth distribution, and estimation of outgoing bandwidth which are relevant in overlay multicast. With reputation algorithms (e.g. [23]), nodes that tend to contribute more obtain higher reputations and may be prioritized when they download data. However, with streaming, the added challenge is that the instantaneous bandwidth demand and supply in the system need to be balanced, and arbitrary discretization is not possible. Thus, there is a need for mechanisms that compute overall instantaneous system resources, and decide on an allocation policy at that instant. In addition, subsidization is needed since some peers are genuinely not capable of donating bandwidth. Similarly, as discussed in Section II-B, the timeliness requirements of broadcasting precludes

the use of tit-for-tat techniques used in applications like BitTorrent [18].

Recently, researchers have begun to investigate incentive mecanisms for overlay multicast. An incentive mechanism that provides service differentiation in peer selection based on relative contribution of peers is proposed in [39]. An approach to regularly rebuild multicast trees and require nodes to only track their first-hand behavior to deny service to freeloaders is considered in [40]. The impact of tree reconstruction on application performance is unclear, and remains to be studied in depth. [41] argues that the p2p streaming structure has a natural inherent incentive for peers to contribute bandwidth to the community, and propose a ”bazaar” framework to leverage this. The theory of mechanism design is applied to the overlay multicast problem in [42]. A key difficulty involves computing the value that a new node introduces to the rest of the system, by comparing the performance of members before and after the node joins. It is unclear how easily such computations may be performed under group dynamics, network dynamics, and the diversity in Internet path performance. Finally, none of these approaches address issues with bandwidth-scarce environments, the need for hosts behind DSL/cable connections to be subsidized by hosts behind higher bandwidth Ethernet connections, prioritization in tree placement, frameworks for application-level degradation, and preventing over-estimation of outgoing access bandwidth of nodes.

IX. Summary

In this paper, we present the design and implementation experience of a contribution-aware overlay broadcasting system targeted at environments where not all nodes can receive the source rate and node contributions are heterogeneous. The system leverages the multi-tree framework, and supports bandwidth distribution policies which enable nodes to see performance commensurate to their contribution. We have conducted a detailed evaluation of the system on PlanetLab using traces from real broadcasts, which helps demonstrate the practical benefits of our heuristics. When compared with a contribution-agnostic system, our results indicate that in bandwidth-scarce environments, our contribution-aware system can improve the 10th-percentile performance of all nodes and high contributors alone by 2-35% and 10-240%, respectively. The system also distributes the available bandwidth more equitably among nodes of similar contributions. For example, in one trace, bandwidth received by 90% of low contributors is within 100 Kbps of the mean. Although nodes in our system suffer tree reductions a little more frequently, they require only 70% as much time to recover. We believe these results are promising and display the potential to extend overlay broadcasting toward ubiquitous deployment in mainstream Internet.

References

[1] Y. Chu, S. G. Rao, and H. Zhang, “A Case for End System Multicast,” in *Proceedings of ACM Sigmetrics*, 2000.

[2] P. Francis, "Yoid: Extending the Internet Multicast Architecture," Apr 2000.

[3] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O. Jr., "Overcast: Reliable Multicasting with an Overlay Network," in *Proceedings of OSDI*, Oct. 2000.

[4] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure," in *Proceedings of USITS*, March 2001.

[5] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points," in *Proceedings of ACM SIGCOMM*, 2004.

[6] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proceedings of ACM SIGCOMM*, Aug. 2002.

[7] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth Content Distribution in Cooperative Environments," in *Proceedings of SOSP*, 2003.

[8] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," in *IEEE Journal on Selected Areas in Communications Vol. 20 No. 8*, Oct 2002.

[9] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proceedings of SOSP*, 2003.

[10] J. Liebeherr and M. Nahas, "Application-layer Multicast with Delaunay Triangulations," in *Proceedings of IEEE Globecom*, Nov. 2001.

[11] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proceedings of NOSSDAV*, May 2002.

[12] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level Multicast using Content-Addressable Networks," in *Proceedings of NGC*, 2001.

[13] W. Wang, D. Helder, S. Jamin, and L. Zhang, "Overlay Optimizations for End-host Multicast," in *Proceedings of NGC*, Oct. 2002.

[14] S. Q. Zhuang, B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Bayeux: An Architecture for Scalable and Fault-Tolerant Wide-Area Data Dissemination," in *Proceedings of NOSSDAV*, Apr. 2001.

[15] Y. Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang, "Early Experience with an Internet Broadcast System Based on Overlay Multicast," in *Proceedings of USENIX*, June 2004.

[16] "Tmesh broadcast system," http://warriors.eecs.umich.edu/tmesh/tmeshv.html.

[17] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "DONet/CoolStreaming: A Data-driven Overlay Network for Live Media Streaming," in *Proceedings of IEEE INFOCOM*, 2005.

[18] B. Cohen, "Incentives build robustness in bittorrent," in *Proceedings of First Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[19] Y. Kulbak and D. Bickson, "The emule protocol specification," in *Technical report TR-2005-03, the Hebrew University of Jerusalem*, 2005.

[20] Y. hua Chu, J. Chuang, and H. Zhang, "A case for taxation in peer-to-peer streaming broadcast," in *Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, 2004.

[21] D. Dutta, A. Goel, R. Govindan, and H. Zhang, "The design of a distributed rating scheme for peer-to-peer systems," in *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[22] S. Buchegger and J. Boudec, "A robust reputation system for p2p and mobile ad-hoc networks," in *Proceedings of the Second Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[23] H. T. Kung and C.-H. Wu, "Differentiated admission for peer-to-peer systems: Incentivizing peers to contribute their resources," in *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[24] V. K. Goyal, "Multiple Description Coding: Compression Meets the Network," *IEEE Signal Processing Magazine, Vol. 18*, pp. 74–93, 2001.

[25] P. Chou, H. Wang, and V. Padmanabhan, "Layered Multiple Description Coding," in *Proceedings of Packet Video Workship*, 2003.

[26] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," in *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, Aug. 2003.

[27] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2003.

[28] Y. Sung, M. Bishop, and S. Rao, "Enabling contribution awareness in an overlay broadcasting system," in *Proceedings of ACM SIGCOMM*, 2006.

[29] "Esm broadcast system," http://esm.cs.cmu.edu/.

[30] A. Ganjam and H. Zhang, "Connectivity Restrictions in Overlay Multicast," in *Proceedings of NOSSDAV*, June 2004.

[31] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "Insights into pplive: A measurement study of a large-scale p2p iptv system," in Proc. Workshop on Internet Protocol TV (IPTV) services over World Wide Web in conjunction with WWW2006, May 2006.

[32] V. Pai, K. Tamilmani, V. Sambamurthy, K. Kumar, and A. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *Proc. The 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2005.

[33] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous unstructured end system multicast," in *Proc. The 14th IEEE International Conference on Network Protocols*, November 2006.

[34] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live p2p streaming approaches," in *Proceedings of IEEE INFOCOM*, May 2007.

[35] F. Wang, Y. Xiong, and J. Liu, "mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast," in *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, June 2007.

[36] R. Ma, S.C.M.Lee, J.C.S.Lui, and D. Yau, "A game theoretic approach to provide incentive and service differentiation in p2p networks," in *Proceedings of ACM Sigmetrics*, June 2004.

[37] M. Adler, R.Kumar, K.Ross, D.Rubenstein, D.Turner, and D. Yao, "Optimal peer selection in a free market peer-resource economy," in *Second Workshop on Economics of Peer-to-Peer Systems, Cambridge, Massachusetts*, June 2004.

[38] S. Kamvar, B.Yang, and H. Garcia-Molina, "Addressing the Non-Cooperation Problem in Competitive P2P Systems," in *First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[39] A.Habib and J. Chuang, "Incentive mechanism for peer-to-peer media streaming," in *Proceedings of the 12th IEEE International Workshop on Quality of Service (IWQoS'04)*, June 2004.

[40] T. Ng, D. Wallach, and P. Druschel, "Incentives-compatible peer-to-peer multicast," in *Proceedings of the Second Workshop on Economics of Peer-to-Peer Systems*, June 2004.

[41] V. Shrivastava and S. Banerjee, "Natural selection in p2p streaming: From the cathedral to the bazaar," in *Proceedings of ACM NOSSDAV, Skamania, WA*, June 2005.

[42] S. Yuen and B.Li, "Strategyproof mechanisms for dynamic multicast tree formation in overlay networks," in *Proceedings of IEEE Infocom*, April 2005.