Enabling Contribution Awareness in an Overlay Broadcasting System

Yu-Wei Sung, Michael Bishop, and Sanjay Rao
Department of Electrical and Computer Engineering
Purdue University
{sungy,bishopma,sanjay}@purdue.edu

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 outgoing 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.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks]: Distributed Systems

General Terms: Algorithms, Design, Experimentation

Keywords: Overlay multicast, Multi-tree, Incentive

1. INTRODUCTION

In the last few years, application-level overlay multicast has emerged as a key alternative to enable broadcasting applications on the Internet. In this scheme, participants in the broadcast self-organize into efficient overlays where video content is disseminated by the broadcast source without support from the network. There has been significant effort in recent years devoted to validating the architecture [7, 10, 13, 18, 21], designing protocols [1, 3, 4, 7, 10, 13, 14, 16, 17, 19, 24, 27], and deploying real systems [6, 23, 25].

Much success with overlay broadcast deployments has been restricted to homogeneous university environments, and broadcasts involving scientific conferences and lectures. For example, a recent work on experience with an overlay broadcasting system [6] indicates substantial success in achieving good performance in university-based environments, but highlights several performance issues when mainstream environments are considered. In this paper, we seek to enable overlay broadcast in environments characterized by two key properties. First, we consider highly heterogeneous environments where hosts make unequal contributions to the overlay. Such heterogeneity may arise due to different node outgoing capabilities (Ethernet vs. DSL) or different willingness to contribute bandwidth resources. For example, [6] reports the fraction of resource-poor hosts (i.e. with low outgoing bandwidth) in several real broadcast deployments range from 43% to 81%. Second, the bandwidth resources contributed by all hosts may be insufficient for everyone to receive the full source rate.

We present the design of an overlay broadcast system targeted at these environments. Our primary goal is to enable hosts to receive different levels of performance based on their contributions while effectively utilizing the bandwidth resources available in the system. To achieve this, our system leverages the multi-tree framework [3] to enable application-level adaptation. In our system, the source delivers data along multiple overlay trees. Each node subscribes to all trees but is only entitled to a subset of them. The number of trees a node is entitled to depends on the amount of bandwidth it contributes. This in turn determines the bandwidth, and consequently quality, it receives. While the multi-tree framework was originally proposed to improve resiliency, our focus is on using multiple trees to enable application-level adaptation and differential treatment.

We enforce bandwidth distribution policies where a node must contribute more than it is entitled to receive. Such policies are shown to better utilize bandwidth of resource-rich hosts and offer better performance to resource-poor hosts than naive bit-for-bit policies [12]. They facilitate the "contribution-aware" policy: the more a node contributes, the more it is entitled to receive. To support this policy 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 [3, 12], we go substantially beyond these works by presenting a comprehensive contribution-aware design and implementation experience on an operational overlay broadcasting system [6].

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, but achieves a 10-fold reduction in average time to recover from these changes for high contributors.

Section 2 discusses our assumptions and motivations in detail. Section 3 presents the design of contribution-aware heuristics. Section 4 describes the broadcasting system our implementation is based on and the process of integrating the multi-tree framework into this system. Section 5 discusses our evaluation methods and metrics. Evaluation results are presented in Section 6. Based on these results, we draw conclusions on our contribution-aware heuristics in heterogeneous, resource-scarce environments in Section 7.

2. ASSUMPTIONS AND MOTIVATIONS

In this section, we describe the assumptions made on user behavior in Section 2.1. Section 2.2 introduces the multi-tree overlay we leverage to achieve differential resource distribution to nodes of diverse contribution levels. Section 2.3 presents a bandwidth distribution policy which we leverage to distribute bandwidth among peers. We discuss the goals of our contribution-aware system design in Section 2.4.

2.1 Broadcast User Model

To simplify our design, we make certain assumptions about the behavior of broadcast participants. Although we believe our conclusions may be generalized past some of these assumptions, we limit our discussion to this user model.

A peer i in the broadcast is capable of both receiving and forwarding data. Without loss of generality, we assume that every peer is capable of receiving the full *source rate* S in the event 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 frequently would be able to receive and forward several times more than S.

We do not assume a homogeneous *forwarding bound*, but consider different levels of *actual contribution*. A peer i's *actual contribution* f_i is bounded by either *ability* or *willingness*. We assume that this *forwarding bound* F_i is determined by user willingness and is never over-estimated by the user (i.e. willingness $\leq$ ability). We assume that F_i is known only to the user and is non-zero – every peer will contribute some bandwidth upon request. We believe this is realistic since all Internet hosts have some upstream bandwidth. f_i may vary over the course of i's stay in the system due to changes in the number of children i supports. Therefore, we target our design to react to users' *actual contribution*.

An important assumption is users are *not strategic*. While we offer incentives to encourage users to increase willingness to contribute, we do not model users who attempt to contribute minimum possible in order to achieve their desired performance. Instead, a peer i only ensures f_i does not exceed F_i when adopting a new child. Therefore, we assume heterogeneity in F_i will be reflected by heterogeneity in f_i.

Lastly, we assume hosts honestly report their f_i, inferred by the number of children they currently support, and the bandwidth received from parents. However, we believe our heuristics can be easily integrated with recent research in distributed auditing and rating of nodes [8, 2, 15] to verify the claimed contribution of nodes. With these assumptions in mind, our goal is to encourage a host i to relax its F_i, particularly under resource-scarce operating environments.

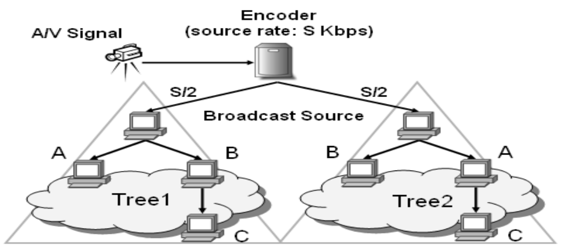

Figure 1: A multi-tree broadcast with two trees.

2.2 Multi-tree-based Data Dissemination

Our system is targeted at regimes where insufficient resources are present in the system for all hosts to receive the full source rate. While we have assumed that all hosts contribute some outgoing bandwidth, we do not assume that everyone can forward the full source rate. In resource-scarce environments, we must also utilize any outgoing bandwidth which is less than the source rate. Therefore, we require a means by which hosts may receive and contribute graduated levels of bandwidth and transition smoothly as available resources change. To realize these goals, we leverage the multi-tree data delivery framework [3, 17]. Although it was first introduced in the context of protocols based on Distributed Hash Table (DHT) [20, 4, 26, 22], the multi-tree framework is not dependent on the use of DHT overlays.

In this framework, participants self-organize into a forest of T trees rooted at a source. The source encodes video with source rate S evenly into T stripes of size S/T, each distributed along one tree. The quality that a host gets depends on the number of stripes that it receives. Typically, a layered codec based on Multiple Description Coding (MDC) [11, 5] 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* to be $degree * T$, which is the maximum number of children a host can support in the tree it contributes. Figure 1 illustrates how 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. tree-degree of 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 [3, 17], we use it as a convenient building block and focus on issues regarding heterogeneity and resource allocation mechanisms.

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 having a degree greater or equal to $1/T$ (i.e. tree-degree ≥ 1) is able to contribute. By setting T properly, we allow resource-poor nodes with limited outgoing bandwidth to contribute, thereby spreading the forwarding load across all peers.

2.3 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.

The multi-tree framework enables us to consider settings where hosts can obtain different video qualities based on the bandwidth they are entitled to, by connecting to a subset of trees at a high priority. Note that the granularity is limited by the number of trees, T, in the forest. Having a larger T enables greater granularity, however one potential cost is an increased overhead due to the MDC coding.

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 will consider a generic cost function of the form proposed in [12] to provide differential distribution:

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

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 left-over 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 3.1.1 describes a distributed way to obtain system-wide estimates such as $\sum_i f_i$ and N.

2.4 Design Criteria

There are some criteria we wish to address in the design of our system. These criteria are targeted to offer good application performance, but also to improve user experiences while preferentially treating high-contributing nodes.

Good Utilization: Given sufficient resource, users should receive bandwidth close to the source rate. While it is not possible for all users to receive the full rate in resource-scarce environments, the bandwidth provided to them should be optimized by making good use of available resources, and there should ideally be no untapped bandwidth. Each node should contribute to the extent of its ability and willingness.

Equitable Distribution: When allocating bandwidth resources among hosts with similar contribution levels, they should receive similar performance.

Differential Distribution: The allocation of bandwidth should favor those which make greater contributions. We also wish to offer some minimum performance to low contributors and give them improved performance if possible.

Stability: Performance should be consistent over time. We expect that a node which sees performance improvements retains them and a node which sees a performance dip recovers quickly.

3. SYSTEM DESIGN

To differentially treat a node based on its contribution in the multi-tree framework, 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 the r_i becomes superfluous. Second, there may be nodes whose *Entitled* bandwidth 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, that is, 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 in that tree and is receiving the data forwarded by 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. We next show how to distribute *Entitled* and *Excess* bandwidth and how nodes are prioritized based on whether they are *Entitled* or *Excess* nodes.

3.1 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. We then determine the number of trees i is required to join to receive r_i. 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.

3.1.1 Distributed System Sampling

Collections of various system-wide parameters, for example, $\sum_i f_i$ and N, are necessary to compute r_i using Equation (1) and for some of our heuristics. 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 every 10 seconds from its children in each tree, processes them, and sends along 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 will extract data from the update with the greatest sequence number.

3.1.2 Computing Number of Entitled Trees

After joining the multi-tree 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 implement some form of smoothing on $T_{i_{sample}}$ to prevent the host from overreacting to peer and network dynamics. There are two transitions that 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 the departure of i's children. Children departures may be considered transient, as another child will be acquired quickly in resource-scarce environments.

Thus, we have implemented a smoothing scheme which tracks immediate increases in $T_{i_{sample}}$, but only gradually responds to decreases. That is, we optimistically assume that reductions are transient and improvements persist. 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 the sample using Equation (2) where $T_{i_{est}}$ is a weighted combination 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. We call this particular smoothing scheme *SmoothDown-Only*. We have also evaluated other possible smoothing methods, and the results are presented in Section 6.5.

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.

3.2 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 bandwidth F_i a node i is willing to forward, it is difficult to determine the amount of theses 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 resource-constrained 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, there exists a possibility that the attempt will 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, our choice of 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 delay 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. We confirm this benefit in Section 6.5.2. 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.

Table 1: Preemption Matrix: Can a disconnected node A displace/preempt a connected node B?

A | B	Contributor (C)	Entitled-NC (ENC)	Excess (EX)
C	By contribution	Yes	Yes
ENC	No	By contribution	Yes
EX	No	No	By # *Excess* trees

3.3 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 *Contributor* is entitled to the tree and forwards its received stripe to its children based on its forwarding bound F_i.

***Entitled Non-Contributor* (*Entitled-NC*)**: An *Entitled-NC* is 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 it may assume a different class in each tree. At any time, a peer joins 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 have implemented a class-based prioritization, summarized in Table 1. 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, *Contributors* may displace non-contributors, whether *Entitled-NC* or *Excess*, and *Entitled-NCs* may displace an *Excess* node. Further, when a parent chooses between two *Entitled* nodes of the same class, the node with a higher contribution level f_i in its *Contributor* tree is chosen. When choosing between two *Excess* nodes, the node with fewer overall connected *Excess* trees is chosen. This preference is only given for a difference of more than one *Excess* tree, since otherwise a displaced node could immediately reclaim its position and destabilize the tree structure. This provides incentives, since those nodes who contribute more will reach the full source rate with fewer connected *Excess* trees, and receive higher priorities over other *Excess* nodes. This type of preemption aims to allow each host to connect to the same number of *Excess* trees.

Finally, to offer more stability/protection to nodes with higher priorities, in the case a node can not find an empty slot, it will preempt, among nodes it knows, the one with the lowest priority. Thus, *Excess* nodes with more connected *Excess* trees than others are most likely to be displaced.

3.4 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 to contribute (i.e. F_i), 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. In Section 6.5.3, we show that using this join mechanism maintains trees in reasonable balance. 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.* These processes repeat until i is entitled to $T_{i_{eff}}$ trees.

4. IMPLEMENTATION

To evaluate our heuristics on a real system, we have chosen to implement them on the ESM Broadcasting System [9]. ESM is a functional overlay broadcast application that has been used in academic conferences and workshops. The code for the ESM client is approximately 43,000 lines. Having this code base available provided a well-structured platform to experiment with our contribution-aware heuristics. The original ESM protocol uses a single-tree overlay to delivery broadcast content. The rest of the section summarizes ESM and describes how we extend it to use multiple overlay trees.

The ESM protocol [6] relies on a gossip-based group membership process to create a single overlay broadcast tree among participating peers. For each node, knowledge about other members is seeded by the source at join-time and augmented through ongoing contact with other nodes. A node periodically sends information about a subset of members it knows to another node picked at random. The member receiving this message will then update its knowledge about other members. When a node i is disconnected from the tree, it starts the parent selection process by probing a random subset of the nodes it knows to inquire whether they could accept it as a child. Each probed node responds with information about whether it has a free space or a preemptable child (a node with a lower priority than i). After a timeout, the node evaluates potential parents, picks the best one based on some configurable metrics and requests addition as a child. An acceptance of this request means a parent node will begin forwarding data to the node, while a rejection causes i to restart the parent selection process.

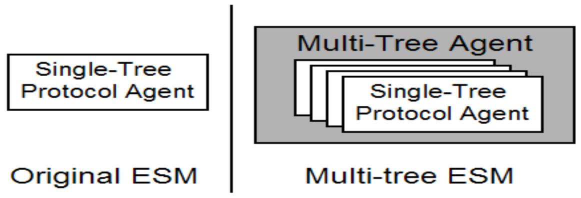

Figure 2: Original vs Multi-tree implementation.

We employ a minimalist approach in adapting the code base to the multi-tree framework. In our multi-tree implementation, we have added 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) as shown in Figure 2. 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 3 into the Multi-tree ESM Broadcasting System.

5. 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 outgoing 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?

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 system refers to multi-tree ESM without any contribution-aware heuristics (e.g. no backoff). When distributing bandwidth, it does not consider nodes' contribution. The only possible preemption is that a *Contributor* can preempt an *Entitled-NC* or *Excess* node. This system is very similar to SplitStream [3] and CoopNet [17].

Cont-Aware: This system refers to multi-tree ESM with our contribution-aware heuristics described in Section 3.

5.1 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 stream, 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 implicit assumption is that the user perceived quality is dictated by the number of trees a node is connected to so each reduction degrades the user perceived streaming quality. We require they 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

Table 2: **Constitution of hosts in a 20-minute segment for each real-world trace.**

Broadcast Event	Avg RI	Low Speed 100Kbps	High Speed 10Mbps	Total Group Size	Peak Group Size
SIGCOMM2002	1.32	34%	66%	88	78
SOSP2003	1.31	47%	53%	95	51
Rally	0.96	73%	27%	401	239
Slashdot	0.87	66%	34%	328	156
GrandChallenge	0.51	88%	12%	281	149

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 be able to reconnect quickly. This may entail 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.

• **Utilization**: Computed as the total bandwidth consumed by all hosts over the total forwarding limits of the hosts at a given time, the metric indicates what fraction of the system resources are being correctly located and leveraged.

5.2 Experimental Methodology

Our study is conducted based on real-world traces obtained from previous operational deployments of the ESM Broadcasting System [6]. Each trace lasts for several hours, and it is clearly not feasible to emulate the full duration. Thus, for each trace, we select a twenty-minute segment with characteristics shown in Table 2. The high-speed and low-speed hosts are generally behind Ethernet and asymmetric DSL/cable connections, respectively. The traces include the join/leave patterns of different nodes, as well as estimates of the outgoing bandwidth of each node. To quantify the resources available in these traces, we introduce *Resource Index* (RI) [6], defined as the total forwarding capacity in the system divided by the bandwidth required for all hosts to receive the full source rate. We classify traces with an RI above one to be resource-rich and resource-scarce otherwise.

The primary trace we use for evaluation is the *Slashdot* trace, which is from a resource-scarce broadcast to an interest-group where the majority of hosts are behind DSL. *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 resource-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 resource-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. Figure 3 shows RI as a function of time for the five trace segments. 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 of 0.25 and 2, respectively.

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

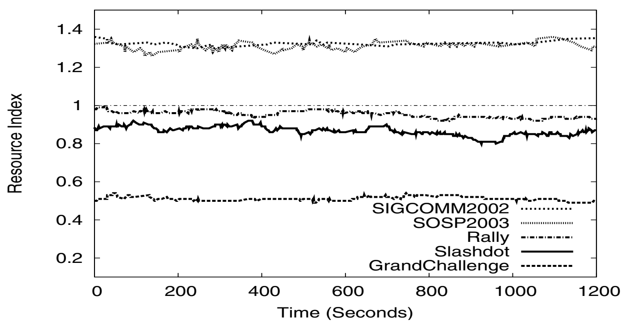

Figure 3: Resource Indices for real-world traces in a 20-minute segment. Traces with an RI above 1 are considered resource-rich and resource-limited otherwise.

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 [6]. 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 in the next section, we filter out hosts which stay for less than 2 minutes to highlight the results for hosts which participate for a reasonable amount of time. We will study the impact of node stay time in Section 6.2.

6. EXPERIMENTAL RESULTS

We begin by showing the behavior of a typical host under the *Cont-Aware* system in a resource-scarce environment using the *Slashdot* trace in Section 6.1. 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 6.2 and Section 6.3, followed by a detailed evaluation of various key design components in Section 6.5. Section 6.6 explores how *Cont-Aware* behaves in different operating environments.

6.1 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.

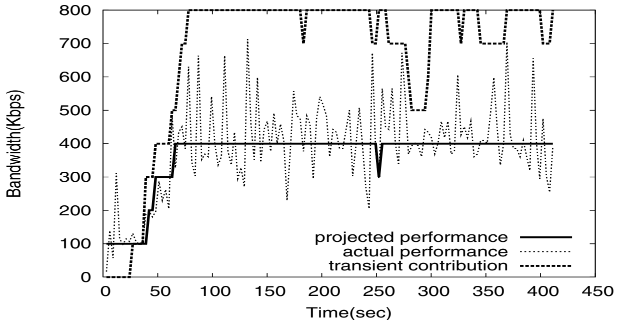

Figure 4: Behavior of a typical high contributor under *Cont-Aware*. The top curve shows the bandwidth contributed, the solid curve shows the *Entitled* bandwidth, and the dashed line shows the actual bandwidth received.

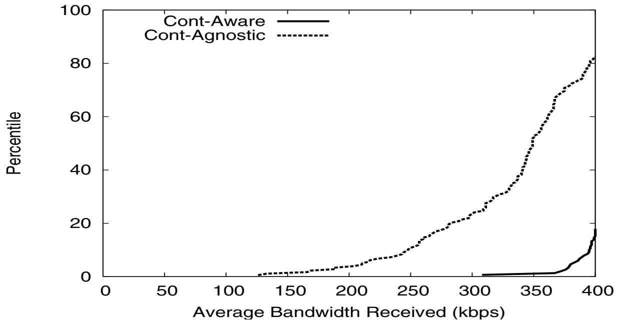

Figure 5: Cumulative distribution of average received bandwidth for high contributors.

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.

6.2 System Performance

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 equal or 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 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*.

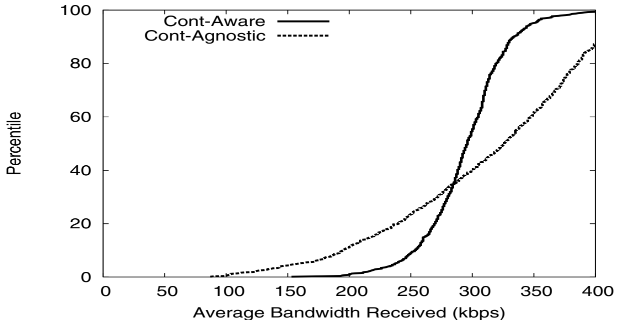

Figure 6: Cumulative distribution of average received bandwidth for low contributors.

While Figure 5 plots the mean bandwidth CDF for high contributors alone, Figure 6 plots the same type of graph, but for low contributors. We see that 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 this. First, we are limited by the granularity imposed by the multi-tree framework, and more equality could result if more trees are created. Second, some clients are limited by the bandwidth near them – further, there are issues related to our experimental artifact as several clients may be mapped to the same PlanetLab machine and compete for incoming bandwidth, causing them to under-perform. 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

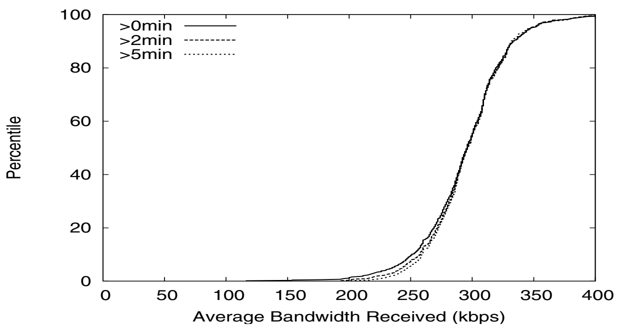

Figure 7: Cumulative distribution of received bandwidth for LC staying for more than 0, 2, and 5 minutes.

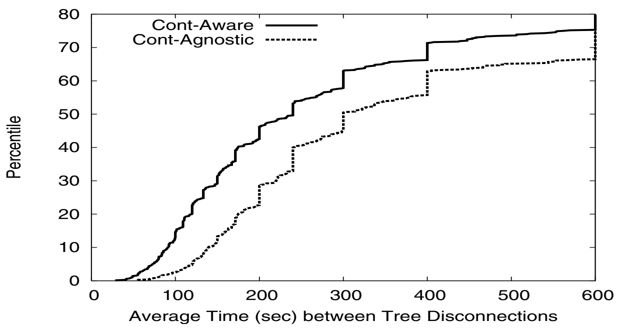

Figure 8: Cumulative distribution of time between tree reductions for all nodes.

connection attempts may work up to a large backoff time, meaning they may never attempt to acquire a parent before they leave the system.

Figure 7 further studies the convergence issues. Each curve corresponds to the performance for low contributors staying longer than X minutes, with X being 0, 2, and 5 minutes. The middle curve is the same as the *Cont-Aware* curve in Figure 6. As we can see, there is a minor improvement in the performance of the tail when longer-lived nodes are considered, however the impact of stay time on performance is negligible overall.

6.3 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 tree. Figure 8 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 3 shows a breakdown of different types of reductions in the number of connected trees. We see that for *Cont-Aware*, only 2.5% of reductions are from two to one

Table 3: Breakdown of different types of reductions in the number of connected trees.

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

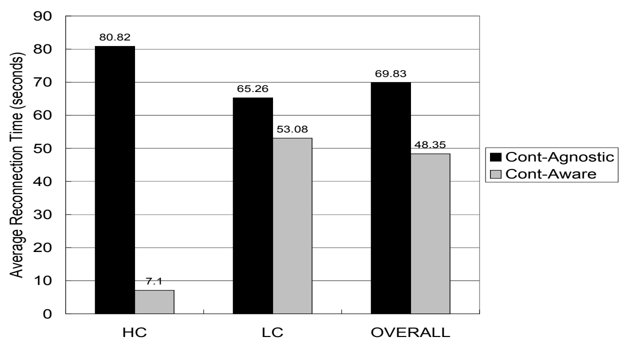

Figure 9: Average post-preemption reconnection time in seconds for nodes in different contribution levels.

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 reduction result from preemptions rather than from parent departures, and *Cont-Aware* allows preempted nodes to reconnect much faster. Figure 9 shows a breakdown of average reconnection time after preemptions. We see that by considering node contribution, the reconnection time for both groups of nodes are reduced, and the reconnection time for high contributors is much shorter than low 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, since *Cont-Aware* establishes finer prioritization levels among nodes, a preempted node can often quickly find a new location in the tree, and the cost of preemption is much cheaper.

To further understand why *Cont-Aware* reduces the reconnection time, Table 4 compares different types of preemptions that may occur based Table 1 and how long the preempted node remains disconnected. In particular, there are three types of preemptions:

- **EN by EN:** An *Entitled* node may be preempted by another *Entitled* node of higher priority. Such type of preemption improves the tree structure by accepting a node which contributes more. The preempted node is also entitled to the tree and should reconnect quickly.
- **EX by EN:** An *Entitled* node may preempt an *Excess* node. This type of preemption is not quickly recovered from, but since the node was not entitled to the tree in which it was preempted, the loss of performance is less significant.
- **EX by EX:** An *Excess* node may be preempted by another *Excess* node. They take place in order to effect equitable performance. An *Excess* node receiving many *Excess* trees will

Table 4: Breakdown of number of preemptions for each preemption type and the average reconnection times following each type of preemption.

Type	HC	LC	Overall	Avg Recon Time
EN by EN	164	307	508	5.95
EX by EN	86	1449	1591	50.51
EX by EX	19	735	784	71.44

be preempted by an excess receiving few, thereby equalizing their bandwidth. If a preempted excess was receiving too many trees, these preemptions should not recover quickly.

In Table 4, we see that the reconnection time following each type of preemption are consistent with our predictions. An *Entitled* node reconnects quickly after a preemption. An *Excess* node preempted by an *Entitled* node does not quickly reconnect. An *Excess* node which is receiving in too many trees and is preempted does not recover quickly at all. We have also observed that most preemptions among high contributors are of the first type, since they are entitled to all four trees most of the time. These preemptions occur when few *Excess* nodes exist in the system. In this case, a *Contributor* sometimes will preempt an *Entitled-NC*. With a reconnection time under 6 seconds, such types of preemption are acceptable. We also see that some high contributor preemptions take place while the high contributor is an *Excess* in a tree; this situation primarily occurs at the beginning of the high contributor's lifetime, before it begins to contribute.

Most preemptions are of the second type, since under resource-scarce environments like *Slashdot*, most *Entitled* nodes are required to connect by preempting other nodes. Since an *Entitled* node prefers to preempt an *Excess* node rather than anther *Entitled* node, the more trees a node is entitled to, the more stable it will be. This ensures nodes are generally connected in their *Entitled* trees for an extended period of time.

6.4 Utilization

In resource-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 does not adversely impact the efficiency of ESM in locating and leveraging the available resources despite with numerous backoffs and preemptions taking place.

6.5 System Dissection

In this section, we evaluate the impact of various heuristics in *Cont-Aware*. We wish to demonstrate the value added by each heuristic and better understand its contribution to the improved performance.

6.5.1 Smoothing Schemes

The goal of smoothing is to enable hosts to experience less frequent reductions in the number of connected trees by not overreacting to global and local transients. In this section, we study three smoothing policies: *No-Smooth*, *Smooth*, and *SmoothDown-Only*, and explain why our heuristics pick *SmoothDown-Only* over the other two schemes to compute the estimated number of *Entitled* trees, $T_{i_{est}}$. Using the notations defined in Section 3.1.2, we first define each policy:

• **No-Smooth:** No smoothing at all (i.e. $T_{i_{est}} = T_{i_{sample}}$), regardless of how abruptly $T_{i_{sample}}$ changes.

• **Smooth:** Always smooth by using Equation (2) to calculate $T_{i_{est}}$ whenever $T_{i_{sample}}$ changes.

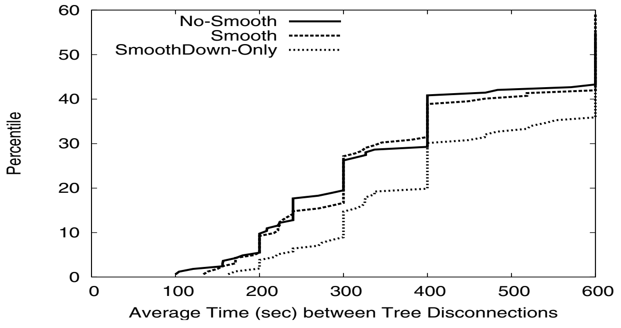

Figure 10: Comparison of time between tree reductions for high contributors under different smoothing schemes.

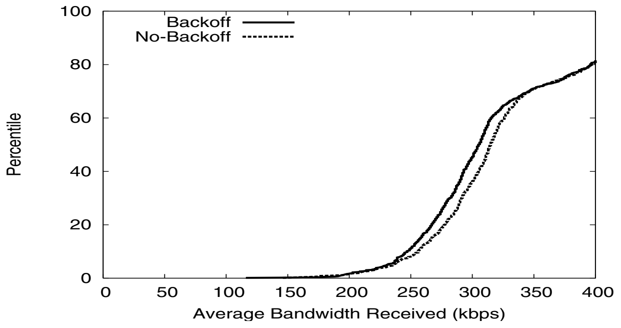

Figure 11: Cumulative distribution of received bandwidth for all nodes under different backoff schemes.

• **SmoothDown-Only:** Smooth by using Equation (2) to calculate $T_{i_{est}}$ only when $T_{i_{sample}}$ decreases.

Our results show that, regardless of the smoothing schemes used, low contributors see similar low time between reductions in the number of connected trees. Figure 10 plots CDF of the average time between reductions in the number of connected trees for high contributors. We see that *No-Smooth* does not perform well because $T_{i_{est}}$ fluctuates with $T_{i_{sample}}$ whereas *Smooth* is as bad because it does not quickly reward nodes whose contributions go up. *SmoothDown-Only* significantly increases the stability for high-contributors over the other two scheme because many decreases in $T_{i_{sample}}$ are caused by transient situations, such as a child departure, in which case another child will be acquired quickly in highly-utilized resource-scarce environments.

6.5.2 Backoff Schemes

In this section, we first justify why we incorporate the backoff mechanism in *Cont-Aware*. Then we investigate why it is beneficial to adding the scaling factor, $T_{i_{excess}}$, in the computation of the backoff timer. We first define three variations of backoff policies:

• **No-Backoff:** No backoff at all. A disconnected *Excess* node immediately attempts to reconnect.

• **Backoff:** When failing to connect, backoff using the timer computed by Equation (3).

• **Backoff w/o $T_{i_{excess}}$:** When failing to connect, backoff using timer computed by Equation (3) with $T_{i_{excess}}$ removed.

Benefit of Backoff: We now examine whether *Excess* nodes

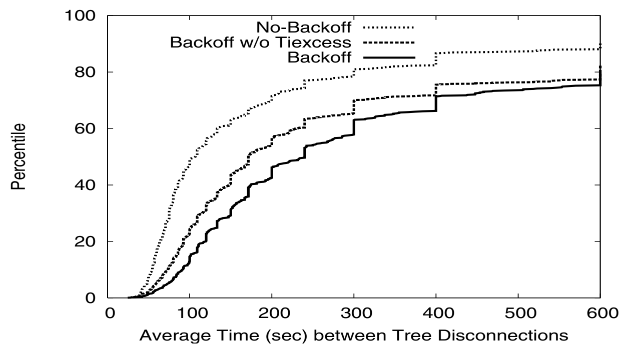

Figure 12: Cumulative distribution of time between tree reductions for all nodes under different backoff schemes.

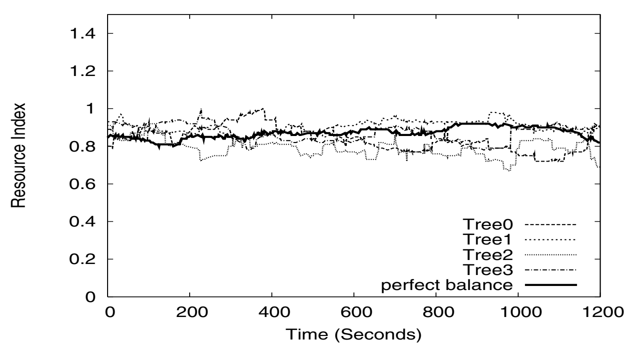

Figure 13: Comparison of resource indices in each tree as compared to perfect load balancing.

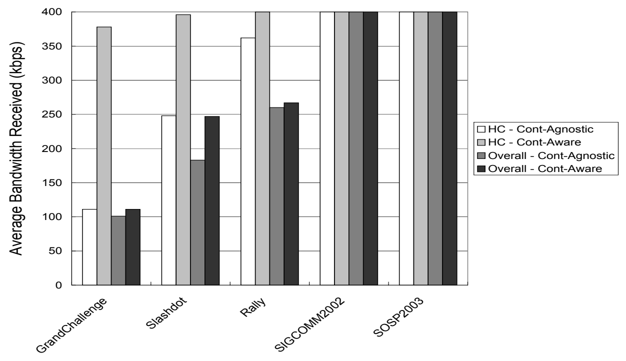

Figure 14: 10th-percentile of received bandwidth for high contributors and all nodes in each trace.

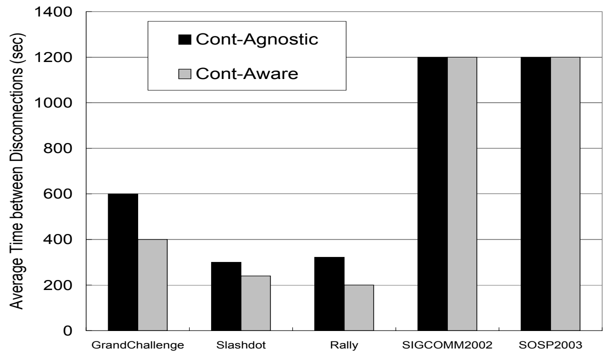

Figure 15: Median of average time between tree reductions for all nodes in each trace.

should backoff when they are unable to connect by comparing two systems: *Backoff* and *No-Backoff*. In Figure 11, we observe that although *No-Backoff* improves the overall performance slightly, Figure 12 shows that *Backoff* significantly improves system stability. Since the environment we consider is resource-scare, *Excess* nodes are very likely be preempted. By trying less frequently to connect to an *Excess* tree unless more resources are available, an *Excess* node is less likely to be quickly preempted, leading to less frequent reductions in the number of connected trees.

Consider $T_{i_{excess}}$ in Backoff Timer: In *Backoff*, in addition to double the backoff timer after each consecutive failure in connection attempts for an *Excess* node, the backoff time is scaled by the number of *Excess* trees the node is currently connected, $T_{i_{excess}}$. Figure 12 compares the stability of *Backoff w/o* $T_{i_{excess}}$ with *Backoff*. We see that *Backoff* leads to a more stable system than *Backoff w/o* $T_{i_{excess}}$ by forcing peers receiving in more *Excess* trees to wait longer before the next reconnection attempt because even if they get connected, they are likely to be immediately preempted by other *Excess* nodes with a lower $T_{i_{excess}}$.

6.5.3 Load Balancing

We believe that choosing to balance the number of *Excess* nodes in each tree would approximately balance the resources in each tree in resource-scarce environments. Figure 13 plots four dashed curves showing the resource index for each tree over time. The solid line represents the ideal RI over time if we have perfect balancing. Although at some instant, the RI for each tree deviates from the ideal curve, the resources each tree have remain close to perfect balancing for the most of the duration. Thus, we conclude that our heuristics maintain trees in reasonable balance in resource-scarce environments.

6.6 Sensitivity to Trace

In this section, we evaluate our contribution-aware heuristics under environments with varying resource levels by using five different traces. We include *Slashdot* among these for comparison to results in previous sections.

Figure 14 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 here. The traces are ordered based on their resource indices, with the lowest RI on the very left. The three traces on the left are resource-scarce whereas the two on the right are resource-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 resource-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. For resource-rich traces, we see similar performance – everyone successfully receives the source rate.

In Figure 15, we examine the sensitivity of the average time between reductions in the number of connected trees to different traces by comparing the median value of all nodes.

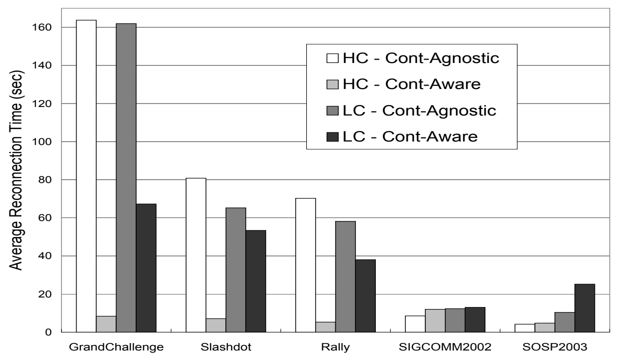

Figure 16: Average post-preemption reconnection time for high contributors and all nodes in each trace.

Notice that for resource-scarce traces, *Cont-Aware* causes nodes to experience reductions more frequently. In resource-rich environments, reductions are infrequent for both as most nodes can connect to empty slots.

Finally, we compare the reconnection time after preemptions in Figure 16. Note that for all resource-scarce environments our heuristics significantly improve reconnection time for both high and low contributors. In resource-rich environments, we offer substantially similar recovery time.

7. SUMMARY

In this paper, we present the design and implementation experience of an overlay broadcasting system targeted at environments where not all nodes can receive the source rate and node contributions are heterogeneous. To incenvitizes nodes to increase their contributions, the system is contribution-aware: it distributes more bandwidth to nodes which contribute more. We have conducted a detailed evaluation of the system on PlanetLab using traces from real broadcasts, which helps demonstrate the benefits of the heuristics we introduce. When compared with contribution-agnostic system, our results indicate that in resource-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.

8. REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] Y. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM Sigmetrics*, June 2000.

[8] 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.

[9] Esm broadcast system. http://esm.cs.cmu.edu/.

[10] P. Francis. Yoid: Extending the Internet Multicast Architecture. Apr 2000.

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

[12] Y. hua Chu, J. Chuang, and H. Zhang. A case for taxation in peer-to-peer streaming broadcast. In *PINS '04: Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, pages 205–212, New York, NY, USA, 2004. ACM Press.

[13] 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 the Fourth Symposium on Operating System Design and Implementation (OSDI)*, Oct. 2000.

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

[15] 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.

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

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

[18] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An Application Level Multicast Infrastructure. In *Proceedings of 3rd Usenix Symposium on Internet Technologies & Systems (USITS)*, March 2001.

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

[20] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[21] 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.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, 2001.

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

[24] W. Wang, D. Helder, S. Jamin, and L. Zhang. Overlay Optimizations for End-host Multicast. In *Proceedings of Fourth International Workshop on Networked Group Communication (NGC)*, Oct. 2002.

[25] 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.

[26] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Wide-area Fault-tolerant Location and Routing. *U. C. Berkeley Technical Report UCB//CSD-01-1141*, Apr 2000.

[27] 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.